VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

3D vizualizace modelů v PL^1

3D vizualization of PL^1 models

2010 Vladimír Matějičný

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli a byli mi po celou dobu oporou, především pak Mgr. Martině Číhalové.

Abstrakt

Predikátová logika prvního řádu dnes nachází velké využití v oblasti matematiky, filosofie, lingvistiky a umělé inteligence. Problémem je však nedostatek kvalitního výukového software. Cílem této práce je vytvoření aplikace, schopné přehledně a jednoduše naučit studenty základům predikátové logiky prvního řádu. Začátek práce je věnován popisu sémantiky predikátové logiky prvního řádu a vymezení podmnožiny formulí, kterou se budeme zabývat. Hlavní náplní textu práce je však především popis vytvářené aplikace na konceptuální úrovni. Tedy jednotlivých funkcí, základních algoritmů a návrh nejdůležitějších částí pomocí jazyka UML. Výsledky této práce budou následně využity v projektu CZ.1.07/2.2.00/07.0217, z tohoto jsou všechny algoritmy popsány v pseudokódu, abychom se oprostili od závislosti na konkrétním programovacím jazyce.

Klíčová slova: PL[1], predikátová logika prvního řádu, Tarski's World, sémantika, formule, interpretační struktura, proměnná, predikát, konstanta, funkční symbol, universum diskursu, individuum, pravdivost v interpretaci, valuace, model formule, průvodce interpretací, generátor formulí

Abstract

First-order logic is now days used in many fields in mathematics, philosophy or artificial intelligence. The problem is that there is not enough quality educational software. The main goal of this thesis is to create an application capable of simply and clearly teaching students basics of first-order logic. The beginning of this paper is dedicated to semantics of first-order logic and defining the subset of formulas which we are going to work with. But the main goal is to describe created application on conceptual level. We will go through main functionality, basic algorithms and design of main parts of the application by means of the UML language. The results of this paper will be used in project CZ.1.07/2.2.00/07.0217. Because of this all of the algorithms are described in pseudocode.

Keywords: PL[1], first-order logic, Tarski's World, semantics, formula, interpretation structure, variable, predicate, constant, function symbol, universum, entity, validity, formula model, interpretation advisor, formula generator

Seznam použitých zkratek a symbolů

PL1	–	Predikátová logika prvního řádu
DRY	–	Do not repeat yourself
UML	–	Unified model language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Predikátová logika prvního řádu nachází velké využití v oblasti matematiky, filosofie, lingvistiky a umělé inteligence. Může být použita pro formalizaci mnoha matematických důkazů nebo vytváření znalostních bází. Dokonce i mnoho věcí z našeho každodenního života může být vyjádřeno predikátovou logikou prvního řádu. Velkého významu se predikátové logice prvního řádu dostává v oblasti umělé inteligence, jelikož teoreticky vše co pomocí ní může být vyjádřeno může být také naprogramováno. Proto je důležité si osvojit alespoň její základy. To ovšem může pro některé studenty představovat problém a to také je důvodem vzniku aplikace, jenž tvoří součást této práce. Konkrétně se naše aplikace bude snažit objasnit studentům problematiku pouze jedné oblasti predikátové logiky prvního řádu - *modely formulí.*

Pro studenty je možná vůbec největším problémem při hledání modelů formulí pochopení, co má vlastně taková formule predikátové logiky první řádu vůbec vyjadřovat. Studenti většinou správně identifikují jednotlivé elementy formule, jako jsou kvantifikátory, predikáty nebo logické spojky. Ovšem celkové pochopení interakcí mezi těmito jednotlivými elementy formule je obtížné, vyžaduje totiž velkou míru abstraktního myšlení. Pokud při řešení jakéhokoliv problému, zapojíme do hry naše abstraktní myšlení a plně tomuto problému nerozumíme, pohybujeme se v tzv. "šedé zóně", kdy jsme sice schopni se po čase dopracovat ke správnému řešení, nicméně se tak děje spíše dílem náhody. Pokud pozměníme určité parametry problému, nemusíme se jeho opětovného řešení vůbec dobrat. Naše aplikace se proto snaží snížit potřebnou míru abstrakce tak, že zobrazuje jednotlivá individua jako tvary v trojrozměrném prostoru. Takto jsou přehledně zachyceny všechny vlastnosti individuí a relace mezi nimi. Nicméně, fakt že se aplikace bude držet pouze jednoho typu universa může nyní vzbuzovat dojem, že takto student bude umět interpretovat pouze formule na universu prostorových tvarů. Toto ovšem není vůbec pravda, neboť nabyté znalosti jsou zcela universální a dají se aplikovat na jakýkoliv typ universa.

1.1 Tarski's World

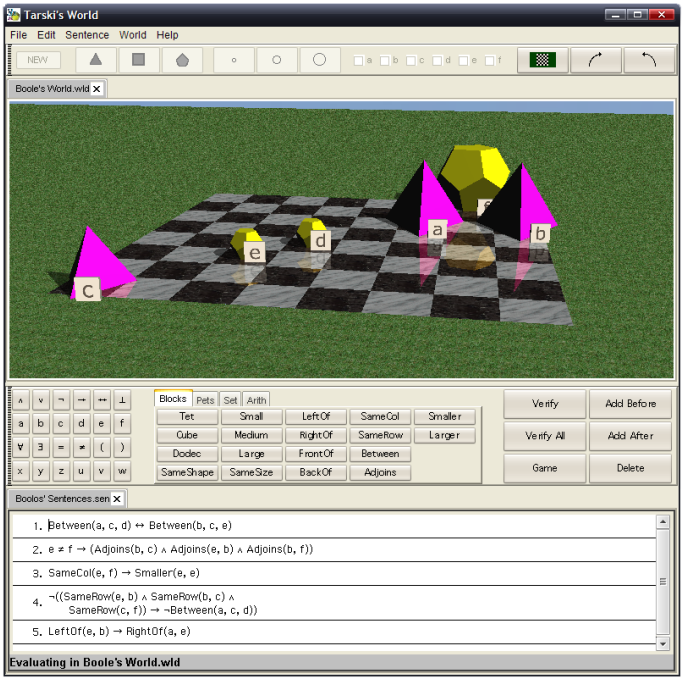
V této oblasti však dosud neexistuje dostatek kvalitního výukového software. K těm kvalitnějším aplikacím patří *Tarski's World*, který je součástí stejnojmenné knihy [1]. Svou funkcionalitou se řadí mezi špičku výukového software, bohužel však neoplývá příliš přívětivým uživatelským prostředím. Na obrázku 1 můžeme vidět, že zobrazování universa tvarů je zde provedeno na šachovnici umístěné v 3D prostoru. Universum je možno složit z celkem tří tvarů(*čtyřstěn, krychle a dvanáctistěn*) různých velikostí odstupňované od *malé*, *střední* po *velkou*. Každý z tvarů universa má již předem danou barvu a není možno ji změnit.

Nalezneme zde také jednoduchý editor formulí, pomocí kterého si můžeme sestavit v podstatě libovolnou formuli, kterou již následně není třeba interpretovat, protože názvy jednotlivých predikátových symbolů přímo korespondují s jejich významem. Tedy kupříkladu predikát *Cube(x)* znamená, že "individuum x je krychlí."Těchto predikátů zde nalezneme celou řadu, namátkou vyberme predikát $SameSize(x, y)$ - "x je stejně

velké jako y"nebo $LeftOf(x, y)$ - "x je nalevo od y". Díky tomuto způsobu pojmenování si student lépe uvědomí význam konkrétního predikátu ve dané formuli, nicméně však neodpovídá způsobu uvedeném v definici 2.1 a proto se v této práci budeme držet klasického pojmenování velkými písmeny abecedy P, Q, R atd. Tímto mechanismem můžeme pro jedno universum sestavit hned několik formulí a postupně postupně ověřovat jejich pravdivost v interpretaci. V Tarski's World ovšem nenalezneme žádné funkční symboly vyjma konstant. Je to zřejmě z důvodu poněkud složitější interpretace pro tento typ universa. Nejedná se však o neřešitelnou situaci, jedna z možností jak lze zde funkční symboly interpretovat je popsána v kapitole 4.3.

Zajímavou funkcí je tzv. *"Game"* mód. V tomto módu se aplikace studenta například dotazuje zda je přesvědčen o pravdivosti formule, kterou vytvořil nebo zda si je jist významem některé z jejích části. Velmi užitečné je dotazování typu *"Jste si opravdu jist, že všechna individua universa jsou krychle?"*, pokud se bude jednat například o formuli $\forall x P(x)$, kde $P(x)$ = *x je krychle*.

Tarski's World se neomezuje pouze na prostorové tvary, umožňuje také universum zaměnit za zvířata nebo ověřovat pravdivost jednoduchých matematických a množinových operací. Nežijeme však v ideálním světě a proto ani tato aplikace není dokonalá, trpí mnoha neduhy, které mohou odradit potencionální studenty.



Obrázek 1: Tarski's World

1.2 Cíle práce

Cílem této práce tedy bude zachování většiny funkcí "Tarského světa,"celkové zjednodušení uživatelského prostředí a přidání nových funkcí, především pak následující:

- Zajištění větší variability universa rozšířením množiny prostorových tvarů a možností přiřadit každému z těchto tvarů určitou barvu z předem definované množiny barev. Dále pak přidání "skutečného"třetího rozměru, tak aby bylo možno s tvary pohybovat po osách x, y a z.
- Aplikace bude obsahovat dva způsoby vytvoření formule. Prvním bude její sestavení pomocí jednoduchého editoru. Druhým bude generátor náhodných formulí, dle daných omezení. Při náhodném generovaní stavíme před studenta úkol vypořádat se s interpretací formule, s níž například ještě nepřišel do styku a tímto může daleko lépe pochopit danou látku.
- Rozšíření možností interpretační struktury přidáním dalších interpretací unárních a binárních predikátů a také zavedením unárních funkčních symbolů.
- Posledním rozšířením bude tzv. "průvodce interpretací."Tento průvodce se bude snažit pomoci studentovi s interpretací formule tím, že celý proces interpretace rozdělí na jednotlivé části, ve kterých bude pomocí jednoduchých otázek a vodítek směrovat studenta ke správné interpretaci formule. Samotné řešení celého problému interpretace ovšem zůstane stále na studentovi samotném.

2 Sémantika predikátové logiky 1. řádu

Predikátová logika 1. řádu(dále jen PL^1) formalizuje úsudky o vlastnostech předmětů a vztazích mezi těmito předměty na pevně dané předmětné oblasti - *universum*. PL^1 je zobecněním výrokové logiky, která nedokáže zachytit vazby mezi vnitřními komponentami elementárních výroků. Abychom však mohli rozhodovat o pravdivosti formulí PL^1 je třeba definovat jazyk, kterým jsou tvořeny a definovat sémantiku jejich formulí - tj. interpretovat speciální symboly jako jsou funkce a predikáty.

Všechny definice v této kapitole jsou čerpány z [2].

2.1 Jazyk PL^1

Definice 2.1 *Jazyk predikátové logiky:*

1. ***Abeceda predikátové logiky*** *je tvořena následujícími skupinami symbolů:*
   (a) ***Logické symboly***
      i. *předmětové (individuové) proměnné:* $x, y, z, \ldots$ *(případně s indexy)*
      ii. *symboly pro spojky:* $\neg, \wedge, \vee, \supset, \equiv$
      iii. *symboly pro kvantifikátory* $\exists, \forall$
      iv. *případně binární predikátový symbol* $=$ *(predikátová logika s rovností)*
   (b) ***Speciální symboly*** *(určují specifiku jazyka)*
      i. *predikátové symboly:* $P, Q, R, \ldots$ *(případně s indexy)*
      ii. *funkční symboly:* $f, g, h, \ldots$ *(případně s indexy)*

      Ke každému funkčnímu a predikátovému symbolu je přiřazeno nezáporné číslo n $(n \geq 0)$*, tzv.****arita****, udávající počet individuových proměnných, které jsou argumenty funkce nebo predikátu.*
   (c) ***Pomocné symboly***
      i. *závorky (), {}, []*
2. ***Gramatika,*** *která udává, jak tvořit:*
   (a) ***termy***
      i. *každý symbol proměnné je term*
      ii. *jsou-li* $t_1, \ldots, t_n (n \geq 0)$ *termy a je-li* f n*-ární funkční symbol, pak výraz* $f(t_1, \ldots, t_n)$ *je term; pro* $n = 0$ *se jedná o nulární funkční symbol, neboli individuovou konstantu (značíme* $a, b, c, \ldots$*)*
      iii. *jen výrazy dle i. a ii. jsou termy*
   (b) ***atomické formule***
      i. *je-li* P n*-ární predikátový symbol a jsou-li* $t_1, \ldots, t_n$ *termy, pak výraz* $P(t_1, \ldots, t_n)$ *je atomická formule*
      ii. *jsou-li* t_1 *a* t_2 *termy, pak výraz* $(t_1 = t_2)$ *je atomická formule*

(c) ***formule***

i. *každá atomická formule je formule*

ii. *je-li výraz A formule, pak $\neg A$ je formule*

iii. *jsou-li výrazy A a B formule, pak výrazy $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $(A \equiv B)$ jsou formule*

iv. *je-li x proměnná a A formule, pak výrazy $\exists x A$ a $\forall x A$ jsou formule*

v. *jen výrazy dle i. – iv. jsou formule*

Pokud je arita funkčního symbolu $n = 0$, pak hovoříme o tzv. *individuované konstantě.* Nejedná se však o pravé logické konstanty, protože podle definice 2.1, stejně jako každý funkční symbol, podléhjí interpretaci. Tyto konstanty značíme jako $a, b, c, \ldots$.

Pro výše definovaný jazyk je charakteristické to, že jediným přípustným typem proměnných jsou *individuové proměnné* (proměnná probíhající určitou předmětnou oblast). Pouze tyto proměnné lze vázat kvantifikátory.

Definice 2.2

Výskyt proměnné x ve formuli A je vázaný, *jestliže je součástí nějaké podformule $\forall x B(x)$ nebo $\exists x B(x)$ formule A.*

Proměnná x je vázaná ve formuli A, *má-li v A vázaný výskyt. Výskyt proměnné x ve formuli A, který není vázaný, nazýváme* ***volný.***

Proměnná x je volná ve formuli A, *má-li v A volný výskyt.*

Formule, v níž každá proměnná má buď všechny výskyty volné nebo všechny výskyty vázané, se nazývá ***formulí s čistými proměnnými.***

Formule se nazývá ***uzavřenou,*** *neobsahuje-li žádnou volnou proměnnou. Formule, která obsahuje aspoň jednu volnou proměnnou se nazývá* ***otevřenou.***

Symbolem $A(x/t)$ označujeme formuli, která vznikne z formule A ***korektní substitucí termu t za proměnnou x***

Pokud vezmeme například v úvahu formuli $\forall x[P(x) \supset Q(x, y)]$, tak proměnná x je vázaná a proměnná y je volná. Jedná se tedy o formuli otevřenou. Volnost a vázanost proměnné se může lišit i na různých místech jejího výskytu. Ve formuli $P(x) \supset \forall x Q(x)$, je první výskyt proměnné x v predikátu P volný, zatímco její druhý výskyt v predikátu Q je vázaný. Příkladem uzavřené formule může být $\forall x \exists y[P(x, y) \wedge Q(x, y)]$, výskyt obou proměnných x a y je v této formuli vázaný.

2.2 Interpretace formulí

Převeďme nyní následující dva jednoduché výroky z přirozeného jazyka do jazyka predikátové logiky:

1. "Všichni hudebníci jsou umělci."

2. "Všechna přirozená čísla dělitelná 3 jsou sudá."

První i druhý výrok můžeme formalizovat následující formulí $\forall x[P(x) \supset Q(x)]$. Vidíme tedy, že oba výroky lze popsat stejnou formulí. Obecně platí, že jedna formule mít nekonečně mnoho modelů. Je evidentní, že první výrok je pravdivý a druhý výrok pravdivý není. Ovšem samotná formule popisující tyto výroky, nám tuto informaci nesděluje. Pokud bychom tedy chtěli rozhodnout o pravdivosti či nepravdivosti dané formule, je třeba ji nejprve interpretovat.

První věc, kterou si musíme při interpretaci formule ujasnit je o čem daná formule hovoří. Musíme tedy vymezit tzv. předmětnou oblast, neboli obor proměnnosti (individuových) proměnných. Tato oblast je *neprázdnou* množinou a nazývá **universum diskursu.** Prvky této množiny jsou **individua.** Dále každému n-árnímu **predikátovému symbolu** přiřadíme určitou n-ární **relaci** (podmnožinu Kartézského součinu) nad universem, která bude vyjadřovat vztahy mezi jednotlivými prvky tohoto universa. Pokud se jedná o unární predikátový symbol(jeho arita je n = 1), přiřadíme mu jistou podmnožinu universa. Stejně tak i n-arnímu **funkčnímu symbolu** přiřadíme určitou n-ární **funkci** nad universem. V neposlední řadě musíme interpretovat i všechny **individuové konstanty**, jímž musíme přiřadit **jedno určité individuum** z universa diskursu. Jakmile je formule takto interpretována, můžeme začít hovořit o její **pravdivosti či nepravdivosti v dané interpretaci.**

Definice 2.3

Interpretace jazyka predikátové logiky 1. řádu *je tato trojice objektů (která je někdy nazývána* ***interpretační struktura****):*

- *Neprázdná množina M, která se nazývá* ***universum diskursu*** *a její prvky jsou* ***individua.***
- *Interpretace funkčních symbolů jazyka, která přiřazuje každému n-árnímu funkčnímu symbolu f určité* ***zobrazení*** $\boldsymbol{f_M : M^n \rightarrow M}$.
- *Interpretace predikátových symbolů jazyka, která přiřazuje každému n-árnímu predikátovému symbolu p jistou n-ární relaci* $\boldsymbol{p_M}$ *nad M, tj.* ***podmnožinu Kartézského součinu*** $\boldsymbol{M^n}$.

Sestavme tedy nyní interpretační struktury k formulím výše uvedených výroků. Připomeňme, že oba výroky jsou formalizovány následující formulí $\forall x[P(x) \supset Q(x)]$.

- Interpretační struktura 1. výroku:
  U = lidé
  P = x je hudebníkem
  Q = x je umělec
- Interpretační struktura 2. výroku:
  U = $\aleph$
  P = čísla dělitelná 3 $\{0, 3, 6, \ldots\}$
  Q = sudá čísla

U takto interpretovaných formulí můžeme nyní rozhodovat o jejich pravdivosti.

2.3 Pravdivost formulí

Definice 2.4

Ohodnocení (valuace) individuových proměnných *je zobrazení e, které každé proměnné x přiřazuje hodnotu $e(x) \in M$ (prvek univerza).* ***Ohodnocení termů*** *e^* indukované ohodnocením proměnných e je induktivně definováno takto:*

- $e^*(x) = e(x)$
- $e^*(f(t_1, t_2, \ldots, t_n)) = f_M(e^*(t_1), e^*(t_2), \ldots, e^*(t_n))$*, kde f_M je funkce přiřazená v dané interpretaci funkčnímu symbolu f*

Definice 2.5

Pravdivost formule A v interpretaci I pro ohodnocení e *individuových proměnných (což značíme $\mid =_I A[e]$ – formule A je pravdivá v I pro e, nebo také A* ***je splněna v I ohodnocením e****), je definována v závislosti na tvaru formule:*

1. *Je-li A atomická formule tvaru:*
   - *(a)* $P(t_1, \ldots, t_n)$*, kde P je predikátový symbol (různý od =) a $t_1, \ldots, tn$ jsou termy, pak $\mid =_I A[e]$, jestliže platí $< e^*(t_1), e^*(t_2), \ldots, e^*(t_n) > \in p_M$, kde p_M je relace přiřazená interpretací I symbolu P – obor pravdivosti P. Tedy individua, která jsou hodnotou termů $t_1, \ldots, t_n$, jsou v relaci p_M.*
   - *(b)* $(t_1 = t_2)$*, pak $\mid =_I A[e]$, jestliže platí $e^*(t_1) = e^*(t_2)$, tj. oba termy jsou realizovány týmž individuem.*
2. *Je-li A složená formule dle bodu 2. c) definice 1.1, tj. je-li tvaru:*
   - *(a)* $\neg B$*, pak $\mid =_I A[e]$ jestliže neplatí $\mid =_I B[e]$*
   - *(b)* $B \wedge C$*, pak $\mid =_I A[e]$, jestliže platí $\mid =_I B[e]$ a $\mid =_I C[e]$*
   - *(c)* $B \vee C$*, pak $\mid =_I A[e]$, jestliže platí $\mid =_I B[e]$ nebo $\mid =_I C[e]$*
   - *(d)* $B \supset C$*, pak $\mid =_I A[e]$, jestliže neplatí $\mid =_I B[e]$ nebo platí $\mid =_I C[e]$*
   - *(e)* $B \equiv C$*, pak $\mid =_I A[e]$, jestliže platí $\mid =_I B[e]$ a $\mid =_I C[e]$, nebo neplatí $\mid =_I B[e]$ a neplatí $\mid =_I B[e]$*
3. *Je-li A formule tvaru:*
   - *(a)* $\forall x B$*, pak $\mid =_I A[e]$, jestliže pro* libovolné *individuum $i \in M$ platí $\mid =_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné x individuum i.*
   - *(b)* $\exists x B$*, pak $\mid =_I A[e]$, jestliže pro* alespoň jedno *individuum $i \in M$ platí $\mid =_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné x individuum i.*

Z definice kvantifikátorů vyplývá, že pokud je universum diskursu konečná množina $M = \{a_1, a_2, \ldots, a_n\}$, pak platí následující ekvivalence formulí:

- $\forall x A(x) \Leftrightarrow A_1(x) \wedge A_2(x) \wedge \ldots \wedge A_n(x)$
- $\exists x A(x) \Leftrightarrow A_1(x) \vee A_2(x) \vee \ldots \vee A_n(x)$

Všeobecný kvantifikátor je tedy zobecněním konjunkce a existenční kvantifikátor je zobecněním disjunkce.Vraťme se nyní k výrokům z 2.2 a s pomocí definic 2.4 a 2.5 vyhodnoťme jejich pravdivost.

Nejprve se zaměřme na první z výroků - *"Všichni hudebníci jsou umělci"*, který byl formalizován formulí $\forall x[P(x) \supset Q(x)]$. Jestliže má být tato formule v dané interpretaci pravdivá, musí pro libovolné individuum z universa diskursu platit $| =_I A[e(x/i)]$. Tato formule je složenou formulí dle definice 2.5 bod 2d. Formule bude tedy v dané interpretaci I pravdivá, jestliže neplatí $| =_I P[e]$ nebo platí $| =_I Q[e]$. Jinými slovy nebude pravdivá pouze v případě, že v universu lidí nalezneme takového člověka, který by byl muzikantem(P), ale nebyl by umělcem(Q). A protože množina muzikantů je podmnožinou umělců, žádného takového muzikanta nenalezneme. Formule je tedy v dané interpretaci pravdivá.

Pokud budeme vycházet z předchozího odstavce a zaměříme se na druhý výrok - *"Všechna přirozená čísla dělitelná 3 jsou sudá."*, dojdeme k závěru že druhá formule nebude pravdivá pokud nalezneme přirozené číslo dělitelné třemi, jenž není sudé. Takové číslo ovšem existuje, například valuace $e(x) = 3$ nebo $e(x) = 21$. Z toho plyne, že formule není v dané interpretaci pravdivá. Vidíme tedy, že pravdivost formule závisí na její interpretaci.

Pravdivostní hodnota formule ovšem nezávisí na hodnotě vázaných proměnných. Obsahuje-li však formule nějaké volné proměnné, můžeme vyhodnotit její pravdivost v interpretaci pouze v **závislosti na ohodnocení** (valuaci) **volných proměnných.** Pravdivostní hodnota formule může být pro různé valuace volné proměnné rozdílná. Pro příklad uveďme tuto formuli:

$$\forall x P(x, y)$$

Jako universum diskursu zvolme množinu přirozených čísel $\aleph$. Binárnímu predikátu P přiřaďme relaci $\{< x, y >, x \geq y\}$. Říkáme tedy, že všechna přirozená čísla x jsou větší nebo rovna určitému číslu y. Přiřadíme-li nyní proměnné y např. číslo 5, bude formule nepravdivá. Naopak pokud přiřadíme-li y číslo 0 bude formule v této interpretaci pravdivá.

Definice 2.6

Formule A je splnitelná v interpretaci I, *jestliže* existuje *ohodnocení e proměnných takové, že platí* $| =_I A[e]$.

Formule A je pravdivá v interpretaci I, *značíme* $| =_I A$*, jestliže pro* všechna *možná ohodnocení e individuových proměnných platí, že* $| =_I A[e]$.

Model formule A *je interpretace I, ve které je A pravdivá.*

Formule A je splnitelná, *jestliže existuje interpretace I, ve které je splněna, tj. jestliže existuje interpretace I a valuace e takové, že* $| =_I A[e]$.

Formule A je tautologií *(logicky pravdivá), značíme* $| = A$*, jestliže je pravdivá v každé interpretaci.*

Formule A je kontradikcí, *jestliže nemá model, tedy neexistuje interpretace I, která by formuli A splňovala.*

Model množiny formulí $\{A_1, \ldots, A_n\}$ *je taková interpretace I, ve které jsou pravdivé všechny formule* $A_1, \ldots, A_n$.

Formule B logicky vyplývá z formulí $A_1, \ldots, A_n$***,*** *značíme* $A_1, \ldots, A_n| = B$ *, jestliže B je pravdivá v každém modelu množiny formulí* $A_1, \ldots, A_n$. *Tedy pro každou interpretaci I, ve které jsou pravdivé formule* $A_1, \ldots, A_n$ *(*$| =_I A_1, \ldots, | =_I A_n$*) platí, že je v ní pravdivá také formule B (*$| =_I B$*).*

Uvažujme následující jednoduchou formuli:

$$\forall x P(f(x), x)$$

Jako universum diskursu zvolme množinu přirozených čísel $\aleph$. Interpretujme predikátový symbol P jako relaci "větší než"$(x > y)$ a funkčnímu symbolu f přiřaďme funkci $\aleph \rightarrow \aleph : f(x) = x^2$. Pro ohodnocení $e_0(x) = 0$ a $e_1(x) = 1$ tato formule není pravdivá. Druhá mocnina čísel 0 a 1 je těmto číslům rovna, tedy nepatří do množiny uspořádaných dvojic relace "větší než."Pro všechna další ohodnocení $e_2(x) = 2, e_3(x) = 3, \ldots$ bude formule pravdivá. Formule je tedy **splnitelná v dané interpretaci I,** ale tato interpretace **není jejím modelem.**

Pozměňme předchozí interpretační strukturu formule a pro funkční symbol f zvolme funkci $\aleph \rightarrow \aleph : f(x) = 2x + 1$. Pro všechna ohodnocení e nyní platí $| =_I A[e]$, protože každé přirozené číslo je menší než jeho dvojnásobek zvětšený o 1. Formule je **v této interpretaci pravdivá** a tato interpretace je zároveň i **modelem dané formule.**

Pokud bychom chtěli vědět, zda má vůbec smysl hledat model formule, musíme zjistit zda se nejedná o **kontradikci.** Jako příklad si uveďme formuli $\forall x[P(x) \wedge \neg P(x)]$, pro niž neexistuje interpretace a valuace, ve které by byla pravdivá. Opakem kontradikce je **tautologie**. Pokud je formule tautologií, tak každá její interpretace je zároveň i jejím modelem. Příkladem takovéto formule je $\forall x[P(x) \vee \neg P(x)]$.

3 Vymezení podmnožiny PL^1

V definici 2.1 jsme vymezili jazyk formulí PL^1. Pouze formule vyhovující definici tohoto jazyka jsou dobře utvořenými formulemi PL^1. Takto utvořených formulí je ovšem nekonečně mnoho a proto je nutno, jak z didaktického tak i z implementačního hlediska, vymezit pouze jejich určitou podmnožinu. Všechna omezení kladená na formule PL^1 můžeme neformálně shrnout do následujících několika bodů:

- Maximální počet kvantifikátorů a různých proměnných je 3, s tím že jsou povoleny i proměnné s volným výskytem. Proměnné budou pojmenovány dle definice, tedy - x, y, x. Délka samotné formule je omezena užitím maximálně 3 logických spojek.
- Maximální počet různých unárních nebo binárních predikátů ve formuli je 4. Pojmenovány jsou opět dle definice - P, Q, R, S
- Povoleny jsou pouze dva unární funkční symboly se jmény f a g. Argumenty těchto funkcí mohou být pouze konstanty nebo proměnné.
- Maximální počet různých konstant(nulárních funkcí) je vzhledem k omezením stanoven na 8. Více konstant nemůže žádná z formulí námi vymezené podmnožiny obsahovat. Pojmenování vychází opět z definice jazyka PL^1 - a, b, c, d, e, a_1, b_1, c_1.

Pro samotnou implementaci je však vhodné všechna tato omezení nějakým způsobem formalizovat. Jako nejvhodnější se jeví použití bezkontextové gramatiky, která nám umožní jednoduše formálně zadefinovat všechna námi kladená omezení na formule PL^1. Jako základ byla použita již dříve vytvořená gramatika z práce *"Interpretace logických formulí PL1 v přirozeném jazyce"* [3]. Výčet všech pravidel gramatiky je následující:

$$
\begin{aligned}
S &\rightarrow QntF \\
Qnt &\rightarrow \epsilon \mid Q \mid QQ \mid QQQ \\
Q &\rightarrow \forall Prom \mid \exists Prom \\
F &\rightarrow Neg(A\,Conn\,A) \mid A \\
A &\rightarrow Neg(A2\,Conn\,A2) \mid A2 \\
A2 &\rightarrow NegPred(Arg) \mid NegPred(Arg, Arg) \\
Arg &\rightarrow VarConst \mid Fce(VarConst) \\
VarConst &\rightarrow Var \mid Const \\
Neg &\rightarrow \epsilon \mid \neg \\
Conn &\rightarrow \wedge \mid \vee \mid \equiv \mid \supset \\
Pred &\rightarrow P \mid Q \mid S \mid R \\
Var &\rightarrow x \mid y \mid z \\
Const &\rightarrow a \mid b \mid c \mid d \mid e \mid a_1 \mid b_1 \mid c_1 \\
Fce &\rightarrow f \mid g
\end{aligned}
$$

Nyní si vysvětleme významy nejdůležitějších neterminálů gramatiky. Za zmínku stojí připomenutí významu symbolu ϵ (epsilon), jenž označuje tzv. *prázdné slovo*. Počáteční neterminál S obsahuje jediné pravidlo $QntF$, které slouží ke generování kvantifikátorů a poté samotného "těla"formule. Význam neterminálu Qnt je zachycen v tabulce 1, pro přehlednost jsou zde vyobrazeny pouze možné kombinace kvantifikátorů bez jimi vázaných proměnných. Neterminály F a A bychom mohli označit jako nejdůležitější z celé gramatiky. Právě ony jsou zodpovědné za výsledný tvar a délku celé formule, jejich význam vzhledem ke tvaru a délce výsledné formule je opět zachycen v tabulce 1. Dalším, neméně důležitým neterminálem je neterminál $A2$, sloužící primárně k výběru unárního či binárního predikátu. Argumentem může být proměnná, konstanta nebo funkce, což je zachyceno neterminálem Arg. Širší význam tohoto neterminálu je taktéž vyobrazen v tabulce 1. Významy ostatních neterminálů jsou na první pohled jasné a není je tedy třeba dále rozvádět.

$$
Qnt \begin{cases} \epsilon \\ \forall \\ \exists \\ \forall\forall \\ \forall\exists \\ \exists\forall \\ \exists\exists \\ \forall\forall\forall \\ \forall\forall\exists \\ \forall\exists\forall \\ \exists\exists\exists \\ \exists\forall\forall \\ \exists\forall\exists \end{cases}
\qquad
F \begin{cases} NegA2 \\ Neg(A2\ Conn\ A2) \\ Neg(A2\ Conn\ Neg(A2\ Conn\ A2)) \\ Neg(Neg(A2\ Conn\ A2)\ Conn\ A2) \\ Neg(Neg(A2\ Conn\ A2)\ Conn\ Neg(A2\ Conn\ A2)) \end{cases}
$$

$$
Arg \begin{cases} Var \\ Const \\ f(Var) \\ f(Const) \\ g(Var) \\ g(Const) \\ Var = \{x, y, z\} \\ Const = \{a, b, c, d, e, a_1, b_1, c_1\} \end{cases}
$$

Tabulka 1: Hlavní neterminály omezující gramatiky

Ukažme si nyní na třech jednoduchých formulích, jakým způsobem bude probíhat kontrola zda formule vyhovuje námi vymezeným pravidlům:

1. $\forall x(Q(x) \supset R(x))$:
   $S \rightarrow QntF \rightarrow QF \rightarrow \forall VarF \rightarrow \forall xF \rightarrow \forall xA \rightarrow \forall xNeg(A2\ Conn\ A2) \rightarrow \forall x(Pred(Arg) \supset Pred(Arg)) \rightarrow \forall x(Q(Arg) \supset R(Arg)) \rightarrow \forall x(Q(Var) \supset R(Var)) \rightarrow \boldsymbol{\forall x(Q(x) \supset R(x))}$

2. $\neg Q(f(a)) \vee R(x, f(g(b)))$:
   $S \rightarrow QntF \rightarrow F \rightarrow Neg(A\ \ Conn\ \ A) \rightarrow (NegPred(Arg) \vee NegPred(Arg, Arg)) \rightarrow$
   $(\neg Q(Arg) \vee R(Arg, Arg)) \rightarrow (\neg Q(Fce(Const)) \vee R(Var, Fce(VarConst)) \rightarrow$
   $\boldsymbol{(\neg Q(f(a)) \vee R(b, f(?))}$

3. $\forall x \exists x(Q(x, y) \supset Q(x))$:
   $S \rightarrow QntF \rightarrow QQF \rightarrow \forall Var \exists VarF \rightarrow \forall x \exists xF \rightarrow \forall x \exists xA \rightarrow \forall x \exists xNeg(A2\ \ B\ \ A2) \rightarrow$
   $\forall x \exists x(NegPred(Arg, Arg) \supset NegPred(Arg)) \rightarrow \forall x \exists x(Q(Arg, Arg) \supset Q(Arg)) \rightarrow$
   $\forall x \exists x(Q(Var, Var) \supset Q(Var)) \rightarrow \boldsymbol{\forall x \exists x(Q(x, y) \supset Q(x))}$

První formule je gramatikou bez problému přijata, protože spadá do námi vymezené podmnožiny. Zato druhá formule správně přijata není, protože argumentem funkce mohou být pouze proměnné nebo konstanty. Nelze tedy přepsat neterminál $VarConst$ na jiný neterminál, jehož pravidla by obsahovala funkční symbol. Zajímavého jevu jsme ale svědky u třetí "formule,"která ačkoliv není vůbec správně utvořenou formulí PL^1 je gramatikou přijata. Příčinnou tohoto jevu je fakt, že tato gramatika slouží pouze k omezení jazyka formulí PL^1, nikoli k jeho definici. Zároveň je tedy třeba zajistit aby formule, byla správně utvořenou formulí PL^1 dle definice 2.1. Aby výše uvedená gramatika byla schopna zajistit i toto, muselo by do ní být přidáno mnoho dalších pravidel a stala by se tak velmi rozsáhlou a nepřehlednou. Nicméně se samotné definici jazyka formulí PL^1 přibližuje natolik, že lze v programovacím jazyce naimplementovat jednoduchý systém, který zajistí i podmínku správně utvořené formule PL^1. Tento systém musí zajistit, aby následující tři typy vstupů **nebyly přijaty:**

1. Vstup, kdy je některým z kvantifikátorů vázána proměnná, která se poté ve formuli vůbec nenachází. Může se například jednat o tyto nesprávně utvořené formule: $\forall x P(a)$, $\forall x \forall y(P(x, a) \vee Q(f(x), a))$, $\forall x \exists y(P(x) \wedge Q(x))$, atp. Je tedy třeba *zkontrolovat zda se vázaná proměnná ve formuli opravdu nachází.*

2. "Formule,"která obsahuje dva kvantifikátory vázající stejnou proměnnou. Například tyto nesprávné formule: $\forall x \forall x P(x)$, $\forall x \exists x P(x) \wedge Q(x)$, atp. Musíme *provést kontrolu, zda kvantifikátory vážou různé proměnné.*

3. Typ "formule,"kde se mohou objevit stejně pojmenované predikáty nebo funkční symboly i přestože mají různou aritu n. Například následující nesprávné formule: $Q(a) \wedge Q(a, b)$, $\forall x P(f(x)) \vee P(f(x), a)$, atp. Zde je nutno *zajistit aby pojmenovaný predikát měl vždy stejnou aritu* n.

4 Funkcionalita aplikace

V této kapitole si na konceptuální úrovni popíšeme základní funkce aplikace, právě tyto funkce musíme brát v potaz při utváření architektury aplikace. Výsledná aplikace bude mít ještě více drobných funkcí než je zde uvedeno, ty jsou však z pohledu konceptuálního návrhu zanedbatelné. Mezi základní funkce řadíme následující:

- **3D zobrazení universa** - uživatel musí mít možnost přidávat a odebírat prvky z univerza, stejně tak musí být umožněn i jejich přesun na jinou pozici. Každé z individuí bude mít čtyři vlastnosti - *tvar*, *velikost*, *barvu* a *pozici*. Tyto jednotlivé vlastnosti jsou blíže popsány v kapitole 4.3

- **Generování a vytváření formulí** - aplikace musí umožnit uživateli vygenerovat náhodné formule, ale také je manuálně sestavit. V obou případech musí být zajištěno, aby všechny formule spadaly do vymezené podmnožiny formulí PL^1, jak je uvedeno v kapitole 3. Uživatelský vstup při vytváření formulí bude vyřešen pomocí vizuálního editoru ne nepodobnému editoru formulí z oblíbené kancelářské aplikace Microsoft Word. Náhodné generování formulí bude založeno na gramatice uvedené v kapitole 3. Implementaci systému náhodného generování formulí si přiblížíme v kapitole 6.2.

- **Interpretace formulí** - zde je třeba zajistit aby uživatel mohl interpretovat veškeré predikátové symboly pomocí připravených interpretací. Musí být také zajištěna interpretace funkčních symbolů. Více o možnostech interpretační struktury v kapitole 4.3.

- **Vyhodnocení pravdivosti formule v interpretaci** - pro každou správně interpretovanou formuli, bude možno pomocí algoritmu uvedeném v kapitole 5.1, zjistit její pravdivost v dané interpretační struktuře. Bude zde možnost ověřit jak jednu, tak i více formulí najednou.

- **Průvodce interpretací** - v kapitole 1.2 jsme se krátce zmínili o průvodci interpretací, jenž by měl uživateli pomoci s úspěšným řešením problému interpretace. Podrobněji si tento systém popíšeme v kapitolách 4.4 a 5.2. V tomto přehledu jej uvádíme jen kvůli zdůraznění, aby bylo zcela jasné, že je jedná o jednu ze zásadních funkcí aplikace.

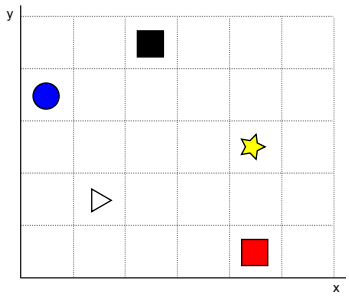
- **Uložení formulí a universa** - samozřejmostí musí být také možnost uložení a opětovné otevření skupiny formulí a universa pro pozdější práci. Bude také umožněno jejich oddělené uložení.

- **Náhodné generování universa** - aplikace bude také umět na základě vstupních parametrů, jako je počet individuí, možné velikosti a barvy tvarů, vygenerovat náhodně sestavené universum.

4.1 3D zobrazení universa

Základem aplikace a více méně i původním cílem této práce je zobrazení universa v trojrozměrném prostoru, v našem případě se jedná o zobrazení v trojrozměrné mřížce. Universum se bude skládat z individuí, jenž budou vyobrazeny jako prostorové tvary. Individua budou mít tyto vlastnosti - *tvar*, *velikost*, *barva* a *pozice*. Pozice každého individua v universu bude unikátní. Vzhledem k přehlednosti a časové náročnosti algoritmů je také vhodné omezit počet možných různých individuí v universu. Stanovme tento počet na **maximálně 20.** Poznamenejme jen že dvě individua se považují za různá, pokud se liší alespoň v jedné ze svých vlastností. Tedy dvě žluté krychle střední velikosti, jsou různá individua protože se nacházejí na různých pozicích. Každému z těchto individuí bude při přidání do mřížky přiřazeno jméno ve tvaru $i_1, i_2, i_3, \ldots, i_n$. Toto pojmenování bude sloužit zejména pro vnitřní potřeby aplikace a také při interpretaci funkčních symbolů.

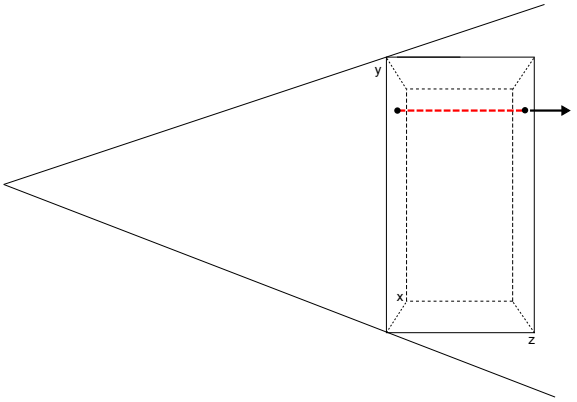
V úvodní kapitole této práce jsme řekli, že individua musí být možno přesouvat na různé pozice v naší 3D mřížce. Zde však narážíme na jeden problém a tím je výběr souřadnic v 3D prostoru pomocí kurzoru myši. Na obrázku 2 můžeme vidět příklad rozmístění tvarů v 2D prostoru. Pozici každého tvaru můžeme bez problému určit při najetí kurzoru myši. Jiná situace ovšem nastává ve 3D prostoru, protože pozice kurzoru myši je pouze 2D bod obsahující souřadnice x a y. Takovýto bod můžeme zobrazit na nekonečně mnoho bodů v 3D prostoru, jak je vidět na obrázku 3. Proto je potřeba zajistit mechanismus, který umožní vybrat souřadnici třetího rozměru. V našem případě je toto zajištěno uzamknutím jedné ze souřadnic y nebo x, tím zajistíme jednoznačný pohyb po rovinách definovaných osami x, y a x, z. Přepínání mezi rovinami je zajištěno pravým tlačítkem myši nebo klávesou Ctrl.



Obrázek 2: Rozmístění tvarů v 2D prostoru

4.2 Interpretace formulí

Aby bylo možno utvořené formule interpretovat, je třeba definovat z jakých individuí se bude skládat universum a jaké interpretace mohou být zvoleny pro predikáty a funkční symboly. Jak jsme se již zmínili, tak universum bude množina prostorových tvarů. Každý z těchto tvarů, nebo chcete-li individuí, bude mít celkem 4 vlastnosti. První vlastností je **pozice** udávaná jako trojice hodnot $< x, y, z >$. Druhou vlastností je **velikost,** která může nabývat těchto tří hodnot - *malá*, *střední* ,*velká*. Další vlastností individua bude jeho



Obrázek 3: Zobrazení 2D bodu ve 3D

barva, ta může být jednou z následujících - *modrá, zelená, červená, žlutá, růžová, oranžová.* A konečně poslední vlastností individua bude jeho **tvar**. Tímto tvarem může být jedno z následujících prosotrových těles:

Prostorová tělesa

- Pravidelný čtyřboký jehlan
- Kužel
- Koule
- Rotační válec
- Krychle
- Pravidelný čtyřboký hranol
- Čtyřstěn
- Osmistěn
- Dvanáctistěn

Dále potřebujeme definovat množinu interpretací pro predikátové symboly. Definujeme celkem osm různých interpretací pro unární predikáty, ty slouží k výběru určité podmnožiny universa. Pro binární predikáty definujeme taktéž osm různých interpretací, které budou naopak vyjadřovat vztahy mezi jednotlivými individui universa. Problematické je ovšem definování pevné množiny interpretací pro funkční symboly s aritou $n \geq 0$. O tomto problému dále níže.

Unární predikáty

<barva> = jedna z 6 barev, <tvar> = jeden z 8 tvarů

- $P(x) =$ x má barvu <barva>
- $P(x) =$ x je <tvar>
- $P(x) =$ x je mnohostěn
- $P(x) =$ x má malou velikost
- $P(x) =$ x má střední velikost
- $P(x) =$ x má velkou velikost
- $P(x) =$ podstava x je kruh
- $P(x) =$ podstava x je čtverec
- $P(x) =$ plášť x je celý nebo částečně tvořen trojúhelníky

Binární predikáty

- $P(x, y) = \{< x, y >,$ x je výše nebo stejně vysoko jako y $\}$
- $P(x, y) = \{< x, y >,$ x je níže nebo stejně nízko jako y $\}$
- $P(x, y) = \{< x, y >,$ x je nalevo od y $\}$
- $P(x, y) = \{< x, y >,$ x je napravo od y $\}$
- $P(x, y) = \{< x, y >,$ x je nad nebo pod y $\}$
- $P(x, y) = \{< x, y >,$ x je vedle y $\}$
- $P(x, y) = \{< x, y >,$ x je blíže nebo stejně blízko jako y $\}$
- $P(x, y) = \{< x, y >,$ x je dále nebo stejně daleko jako y $\}$
- $P(x, y) = \{< x, y >,$ x je větší nebo stejně velké jako y $\}$
- $P(x, y) = \{< x, y >,$ x je menší nebo stejně velké jako y $\}$
- $P(x, y) = \{< x, y >,$ x je stejně velké jako y $\}$
- $P(x, y) = \{< x, y >,$ x je stejně vysoko jako y $\}$
- $P(x, y) = \{< x, y >,$ x má stejnou barvu jako y $\}$
- $P(x, y) = \{< x, y >,$ x má stejný tvar jako y $\}$

Funkční symboly

Funkční symboly v PL[1] musí být definovány na homogenním universu. To však v našem universu prostorových tvarů představuje problém. Představme si situaci, kdy by vstupem interpretace funkčního symbolu $f(x)$ byla některá z barev a výstupem by byla červená barva. Touto interpretací bychom ovšem narušili homogenitu universa, protože naše universum není složeno z prostorových tvarů a barev, ale pouze z prostorových tvarů. Se stejným problémem bychom se potýkali, pokud bychom pomocí funkčního symbolu chtěli měnit kteroukoliv z vlastností individua. Proto asi jedinou schůdnou a logicky čistou cestou, jak vyřešit interpretaci funkčních symbolů s aritou $n > 0$, je jejich definování přímo tabulkou hodnot.

V aplikaci je tedy třeba zabudovat mechanismus, který umožní uživateli sestavit zobrazení jednotlivých individuí $i_1, i_2, i_3, \ldots, i_n$. Systém přitom musí zajistit jednoznačnost zobrazení zprava, jinými slovy individuum $i_x, x \in \{0, \ldots, n\}$ se musí zobrazit právě na jedno individuum $i_y, y \in \{0, \ldots, n\}$. Musí být zajištěna také totalita funkce - funkční hodnota tedy musí být definována pro celé universum. Vzhledem k tomu, že uživatel aplikace bude nucen definovat celou funkci ručně, zůstaneme pouze u unárních funkcí. Interpretace binárních funkcí pro universum s více individui by byla již neúměrně složitá. Interpretace funkčních symbolů s aritou $n = 0$ probíhá jednoduchým spojením určitého funkčního symbolu s vybraným individuem. Příklad definice jednoduché unární funkce představující zobrazení $f : U \to U$ pro universum $U = \{i_1, i_2, i_3, i_4\}$ můžeme vidět v následující tabulce:

U	$\to$	U
i_1	$\to$	i_2
i_2	$\to$	i_3
i_3	$\to$	i_2
i_4	$\to$	i_1

Tabulka 2: Příklad definice funkčního symbolu

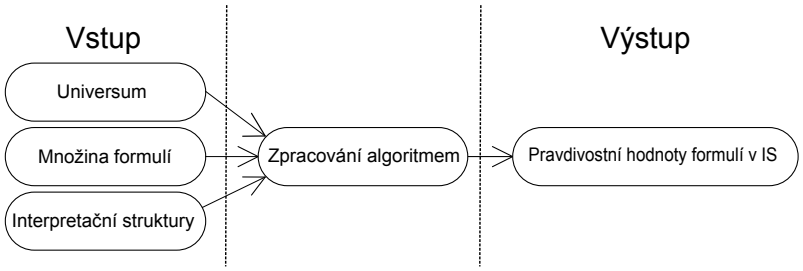
Nesmíme také opomenout fakt, že v situaci, kdy uživatel přidá nebo odebere individuum universa, musí aplikace provést kontrolu, zda tato změna neporušila interpretaci zobrazení, případně aktualizovat interpretace. Pokud se bavíme o zobrazeních s aritou $n = 0$, pak při odebrání individua z universa, nesmíme zapomenout odebrat toto individuum i ze všech interpretací zobrazení, pokud se v některých nachází. U zobrazení s aritou $n = 1$, musí aplikace uživatele upozornit, že přidáním nového individua do universa byla porušena totalita těchto zobrazení a je nutno je opravit. Při odstranění individua z universa je naopak třeba provést kontrolu, zda je funkční hodnota stále definovaná pro celé universum.

4.3 Pravdivost formule v interpretaci

Nyní se jen krátce zmíníme o procesu vyhodnocení pravdivosti formulí v interpretaci. Situace je znázorněna na obrázku 7. Na začátku každé operace vyhodnocení pravdivosti

bude na vstupu universum prostorových tvarů. Universum je již předem vytvořeno, načtením ze souboru nebo manuálně, a je pro všechny formule společné. Proto ho pro názornost uvádíme na vstupu samostatně, nikoli jako součást interpretační struktury. Dále je pak na vstupu skupina formulí, spadající do námi vymezené podmnožiny PL^1 a na závěr interpretační struktura dávající význam jednotlivým formulím. Tato interpretační struktura je vytvořena kombinací konstant, funkcí a již připravených predikátových symbolů. Uživatel musí před začátkem procesu vyhodnocení pravdivosti dle výše popsaného postupu interpretovat všechny funkční symboly a predikáty.

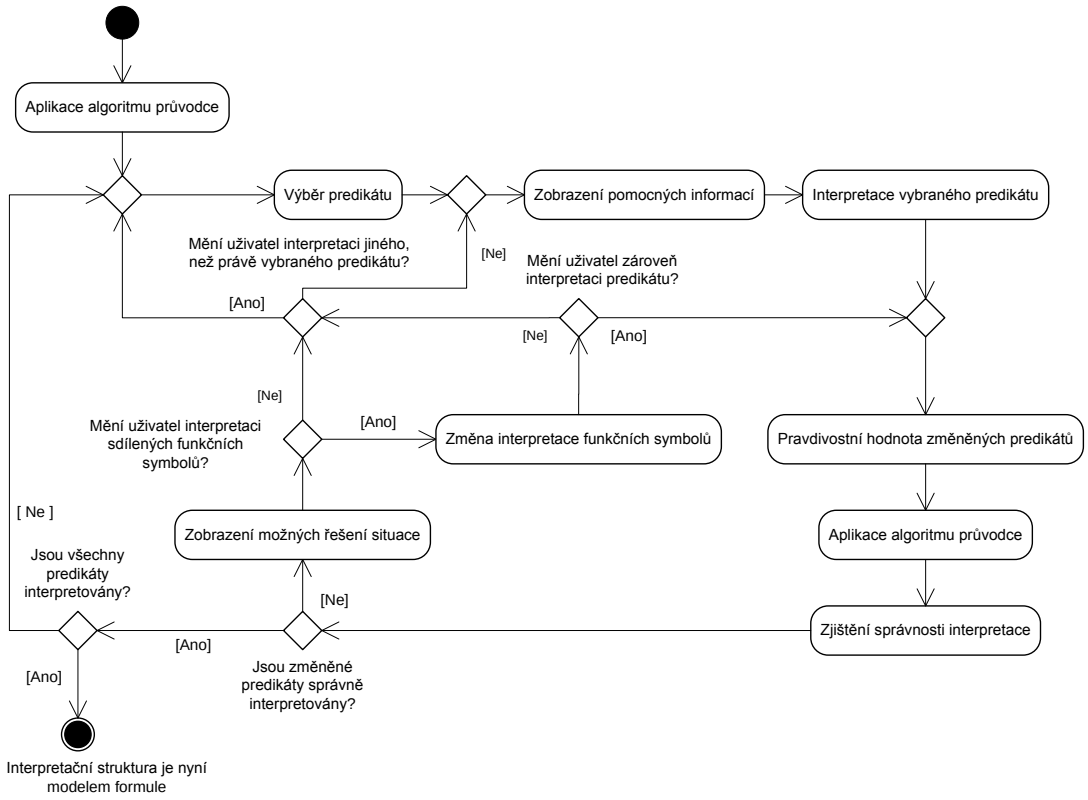
Pokud některá z formulí nebude takto správně interpretována, bude přeskočena a její pravdivost se nebude zjišťovat. Uživateli bychom ovšem měli sdělit, se kterými formulemi se během vyhodnocování pravdivosti nepracovalo. Poté se již můžeme přistoupit k výpočtu a ověření modelů jednotlivých formulí pomocí algoritmu uvedeného v kapitole 5.1. Na výstupu musí být poté zcela jasno, které interpretace formulí jsou pro dané universum pravdivé a které nikoli.



Obrázek 4: Vstupy a výstupy při vyhodnocování pravdivosti formulí

4.4 Průvodce interpretací

V poslední části této kapitoly o funkcionalitě naší aplikace si popíšeme systém průvodce interpretací. Jak jsme již naznačili, tento systém by měl především usnadnit pochopení interpretace formulí PL^1. Na obrázku 5 jsou pomocí UML stavového diagramu zobrazeny jednotlivé fáze průvodce. Všechny tyto fáze si nyní postupně rozebereme, abychom měli bližší představu o jejich rolích v celém systému. Některé fáze si navíc ještě dovysvětlíme na této formuli - $\forall x \forall y[(P(x, f(y)) \supset Q(f(a))) \vee R(f(a))]$ *. Jako universum, na kterém budeme jednotlivé fáze průvodce znázorňovat však nebudeme volit množinu prostorových tvarů. Pokud bychom zde toto universum použili, bylo by potřeba nadefinovat určité množství individuí, což může být bez jejich vizuálního zobrazení velmi nepřehledné. Proto v zájmu zjednodušení a zpřehlednění zvolíme universum přirozených čísel. V samotné aplikaci bude stále použito universum prostorových tvarů a platné budou také všechny níže popsané fáze průvodce.



Obrázek 5: Fáze průvodce interpretací

4.4.1 Výběr predikátu

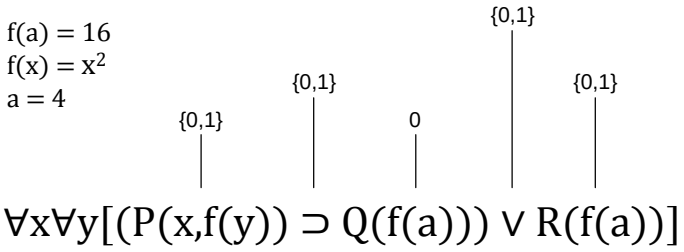
V této fázi si musí uživatel vybrat predikát, který bude chtít dále interpretovat. Uživatel bude nejprve interpretovat predikát a až poté funkční symboly. Pokud byl průvodce právě spuštěn, je nutno na neinterpretovanou formuli ještě aplikovat algoritmus průvodce 4.4.5, tak bychom dostali určitý výchozí bod, od kterého může poté průvodce pokračovat dále. Samotný výběr predikátu nebude probíhat automaticky, bude zcela ponechán v rukou uživatele. Tímto do systému nezanášíme žádné předvídatelné chování ani nevzbuzujeme u uživatele pocit, že pořadí v jakém by aplikace automaticky vybírala predikáty pro interpretaci je to jediné správné. Pro další fáze budeme předpokládat, že uživatel zvolil z naší vzorové formule* predikát $P(x, f(y))$.

4.4.2 Zobrazení pomocných informací

Dále následuje zobrazení pomocných informací. Tato fáze je spouštěna vždy před interpretací predikátu, ať už se jedná o dříve chybně interpretovaný či dosud neinterpretovaný predikát. Průvodce si zde klade za úkol přehledně poskytnout uživateli informace o již interpretovaných predikátech, pokud takové již ve formuli jsou, funkčních symbolech a o logických spojkách formule. Zároveň průvodce uživatele upozorní na možná

úskalí spojená s interpretací jím vybraného predikátu. Celou tuto fázi můžeme rozdělit na následující tři funkce:

- První z funkcí bude uživateli zobrazovat informace o již interpretovaných predikátech a funkčních symbolech. Jako jediná zůstane uživateli zobrazena během všech ostatních fází průvodce. Aktualizována bude postupně na základě změn pravdivostních hodnot predikátů a interpretací funkčních symbolů. Funkce bude také přehledně zobrazovat výstupy funkčních symbolů na jednotlivých místech formule. Předpokládejme nyní, že uživatel již v předchozí iteraci interpretoval predikát vzorové formule* Q jako *x je liché číslo*, funkci f jako $f(x) = x^2$ a nakonec konstantu a jako číslo 4.Zvolená interpretace způsobila, že predikát byl vyhodnocen jako nepravdivý. Uživateli budou tedy na základě těchto skutečností zobrazeny informace o pravdivostní hodnotě predikátů vzhledem ke kvantifikátorům[1] a informace o výstupech jednotlivých funkčních symbolů ve tvaru - $Q(f(a)) = 0, f(a) = 16, f(y) = y^2, a = 4$. Jak bude poté vše vypadat ve spojení s formulí můžeme vidět na obrázku 6.



Obrázek 6: Informace o interpretovaných predikátech a funkčních symbolech

- Další funkcí bude zobrazení informací o logické spojce. Tato funkce již bude závislá na aktuálně vybraném predikátu a její výsledek bude uživateli zobrazen po celou dobu interpretace vybraného predikátu. Smyslem této funkce je pouze zobrazení možných pravdivostních hodnot, kterých může predikát nabývat ve vztahu k logické spojce, tak aby zvolená IS byla modelem formule. Jak bude s touto informací naloženo již záleží na uživateli. Pokud vybraný predikát nebude operandem žádné logické spojky, pak bude tato funkce jednoduše přeskočena. V tabulce 3 můžeme vidět jak bude zhruba vypadat výstup této funkce na uživatelské rozhraní. Interpretace predikátu Q a funkčních symbolů f a a, je stejná jako v předchozím bodě. Jestliže se bude predikát vyskytovat na více místech formule, mluvíme především o formulích s dvěma a více logickými spojkami, pak bude tato tabulka zobrazena pro všechna místa výskytu predikátu.

- Úkolem poslední z pomocných funkcí je upozornit uživatele na možná úskalí spojená s interpretací jím vybraného predikátu. Především máme na mysli situace, kdy

[1] Pravdivostní hodnoty uvádíme v číselném tvaru: 1 - pravda, 0 - nepravda, {0,1} - pravda nebo nepravda

$P(x, f(y))$	$Q(x, f(a))$	$\supset$
0	0	1
1	0	0

Tabulka 3: Zobrazení pomocných informací o logické spojce

je například funkce argumentem více predikátů, tak jak se děje v naší vzorové formuli. Mezi další případy, na které budeme upozorňovat je výskyt vybraného predikátu na více místech ve formuli a nesmíme samozřejmě zapomenout na situaci, kdy je některá z konstant argumentem více predikátů nebo funkcí. Dále je také vhodné uživatele upozorňovat na fakt, že negace predikátu neznamená negaci jeho výsledné pravdivostní hodnoty, ale negaci jeho významu. V kontextu vybraného predikátu budeme uživatele upozorňovat tímto způsobem:

Při interpretaci berte na vědomí tyto skutečnosti:

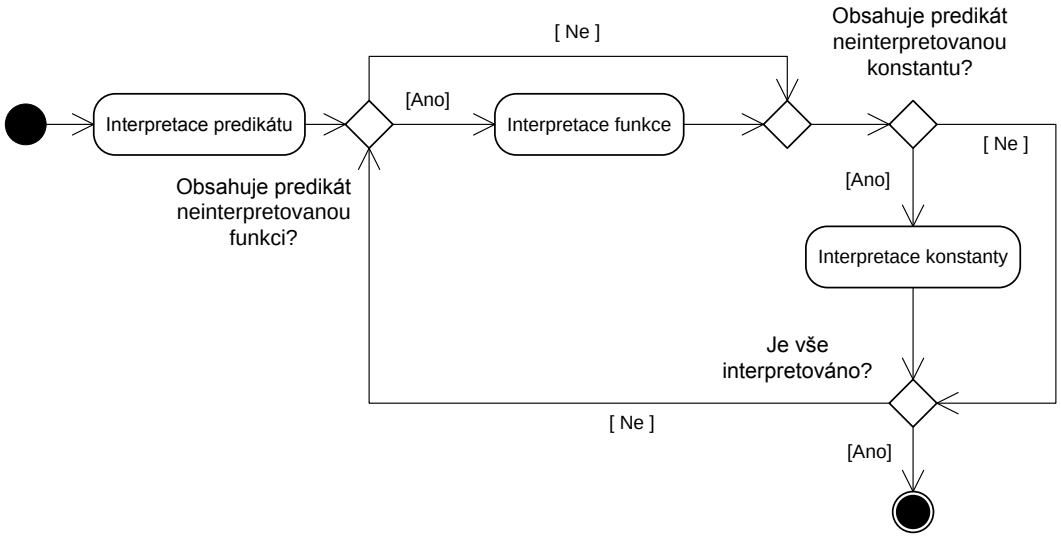
- Predikát ≪*název*≫ se vyskytuje na více místech formule.
- Funkce ≪*název*≫ je také argumentem predikátu(ů) ≪*názvy predikátů*≫.
- Konstanta ≪*název*≫ je také argumentem predikátu(ů) ≪*názvy predikátů*≫.
- Konstanta ≪*název*≫ je také argumentem funkce(í) ≪*názvy funkcí*≫.
- Negace predikátu znamená negaci jeho významu.

4.4.3 Interpretace predikátu

V této fázi má uživatel již dostatek informací, aby mohl interpretovat jím vybraný predikát. Jak celá interpretace probíhá můžeme vidět na UML diagramu z obrázku 7. Z diagramu je jasné, že pokud je argumentem vybraného predikátu již interpretovaná funkce či konstanta, tak uživatel nyní interpretaci těchto symbolů nemění. Toto omezení je zde zavedeno, aby uživatel v této fázi nemohl pozměněním interpretace funkčního symbolu, změnit pravdivostní hodnotu jiného predikátu. Oprava dříve již interpretovaných funkčních symbolů probíhá v samostatné fázi 4.4.8. Jestliže je ovšem argumentem predikátu dosud neinterpretovaný funkční symbol, pak je jeho interpretace samozřejmě umožněna. Poznamenejme ještě, že uživatel se do této fáze vrací i v případě změny interpretace predikátu.

4.4.4 Pravdivostní změněných hodnota predikátu

Jestliže již máme interpretován vybraný predikát i všechny funkční symboly, jenž jsou jeho argumenty, pak dalším logickým krokem bude zjištění pravdivostní hodnoty tohoto predikátu v interpretaci. Vyhodnocení provádíme nezávisle na zbytku formule, pouze s přihlédnutím ke kvantifikátorům. Pro zjištění pravdivostní hodnoty predikátu můžeme využit část algoritmu popsaného v kapitole 5.1. Konkrétně nás budou zajímat implementační funkce *VyhodnoťPredikátJednaProměnná*, *VyhodnoťPredikátDvěProměnné* a také zmíněný postup pro pravdivostní hodnotu predikátu, který neobsahuje žádné proměnné.



Obrázek 7: Interpretace predikátu v průvodci interpretací

Zjišťovat budeme pravdivostní hodnotu nejen aktuálně vybraného predikátu, ale také pravdivostní hodnoty všech predikátů u kterých došlo ke změně jejich pravdivostní hodnoty v důsledku změny interpretace některého z funkčních symbolů. Při zjišťování pravdivostních hodnot nesmíme zapomenout na fakt, že negace predikátu neznamená negaci jeho pravdivostní hodnoty, ale negaci jeho významu.

4.4.5 Aplikace algoritmu průvodce

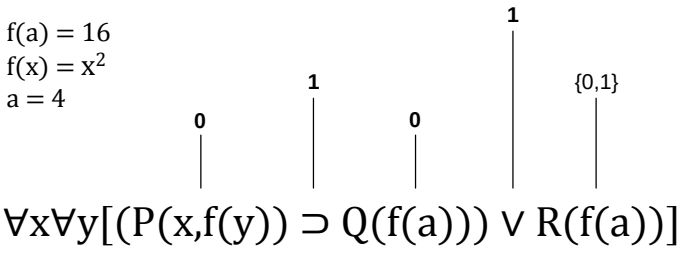
Následující fází průvodce je aplikování uvedeného v kapitole 5.2 na právě interpretovanou formuli. Úkolem algoritmu průvodce není nic jiného, než na základě již známých pravdivostn-ích hodnot predikátů a znalosti struktury formule zjistit, jakých pravdivostních hodnot mohou nabývat všechny predikáty,tak aby celá formule byla pravdivá, a tedy interpretační struktura byla jejím modelem. Protože v této fázi jsou již známy pravdivostní hodnoty všech doposud interpretovaných predikátů, můžeme si dovolit s celou formulí PL[1] zacházet, jako by se jednalo o formuli výrokové logiky a pracovat tedy pouze s pravdivostními hodnotami, negacemi a logickými spojkami. Podrobnější vysvětlení algoritmu si necháme pro kapitolu 5.2.

4.4.6 Zjištění správnosti interpretace

Předpokládejme nyní, že v naší vzorové formuli jsou interpretace predikátu $Q(f(a))$ a $P(x, f(y))$ vyhodnoceny jako nepravdivé a aktuálně vybraným predikátem je $P(x, f(y))$. Pokud budeme tedy vycházet z předchozí fáze, kdy jsme na formuli aplikovali algoritmus průvodce, tak můžeme okamžitě usoudit, že interpretace predikátu $P(x, f(y))$ byla v kontextu naší vzorové formule správná, protože celá implikace nabyla kladné prav-

divostní hodnoty a tato hodnota patří do přístupné množiny správných řešení zjištěné algoritmem průvodce. Jako bonus již v této chvíli víme, že ať už bude interpretace predikátu R jakákoliv, formule bude v dané interpretační struktuře vždy pravdivá. Pokud byla zvolená interpretace vybraného predikátu správná a ve formuli se již nevyskytují žádné další neinterpretované predikáty, pak je celý průvodce ukončen a interpretační struktura je nyní modelem formule. Jestliže se ve formuli nacházejí ještě další neinterpretované predikáty, tak se průvodce vrací do první fáze "Výběr predkátu"4.4.1.

Otázkou ovšem zůstává co s nesprávně interpretovaným predikátem. První věcí, kterou musí průvodce udělat, pokud tato situace nastane, je zobrazit uživateli, které predikáty jsou ve skutečnosti nesprávně interpretovány. Musíme si totiž uvědomit, že například změna interpretace funkčního symbolu $f(x)$ neovlivní pouze aktuálně vybraný predikát $P(x, f(y))$. Jakmile byly uživateli zobrazeny informace o tom na kterých místech formule se vyskytují chyby, průvodce se přesune do fáze 4.4.7, kde nabídne uživateli možná řešení vzniklé situace. Informace o chybných interpretacích zůstanou zobrazeny i v této následující fázi. V této fázi také nesmíme zapomenout aktualizovat pomocné informace zobrazované uživateli. Aktualizované pomocné informace můžeme vidět na obrázku 8.



Obrázek 8: Změna informací o interpretaci

4.4.7 Zobrazení možných řešení situace

Upozorněme, že nesprávnou interpretací je myšlena taková interpretace, která způsobí, že daná IS již nemůže být modelem formule. Pokud byla interpretace uživatelem vybraného predikátu vyhodnocena v kontextu formule jako nesprávná, máme v závislosti na typu formule dvě možné alternativy jak dále postupovat. Uživateli nabídneme tyto možnosti, jak vyřešit nesprávnou interpretaci predikátu:

- První možností je změna interpretace funkčních symbolů. Tato možnost připadá v úvahu pouze tehdy pokud se ve formuli nějaké funkční symboly vůbec nalézají. Pokud ano, nabídneme uživateli možnost změny jejich interpretace. Tuto alternativu rozebereme v následujícím bodě 4.4.8.
- Druhou možností, nezávislou na typu formule, je změna interpretace predikátu. Uživateli bude nabídnuta možnost změnit interpretaci aktuálně vybraného predikátu, tímto se přesuneme do bodu 4.4.3, nebo bude moci změnit interpretaci

kteréhokoliv jiného predikátu formule, z čehož plyne, že se průvodce přesune do fáze 4.4.1. Tato druhá možnost při výběru změny interpretace predikátu je zcela zásadní v případě, kdy se uživatel zároveň rozhodne změnit interpretaci funkčního symbolu, která ovšem vyústí v nesprávnou interpretaci některého z predikátů, jehož argumentem je měněný funkční symbol. Avšak může nastat situace, kdy změnou interpretace, těchto nyní nesprávně interpretovaných predikátů, lze opět dosáhnout stavu, kdy je jejich interpretace považována za správnou.

4.4.8 Změna interpretace funkčních symbolů

Pokud se uživatel rozhodne opravit nesprávnou interpretaci predikátu změnou interpretace jednoho nebo více funkčních symbolů, bude při každé interpretaci průvodcem upozorněn na možné problémy spojené se změnou interpretace jím vybraným funkčním symbolem v souladu s upozorněními uvedenými v 4.4.2. Po každé dokončené interpretaci bude uživateli zároveň nabídnuta možnost, změnit interpretaci dalšího funkčního symbolu, pokud je formule obsahuje nebo změnit interpretaci některého z predikátů. V případě, že se uživatel rozhodne již žádnou další interpretaci neměnit, průvodce se rovnou přesune do bodu 4.4.4 a opět vyhodnotí zda byly aktuální interpretace správné či nikoliv. Jestliže se ovšem rozhodne ještě pro změnu interpretace predikátů, průvodce v závislosti zda uživatel chce měnit interpretaci posledního vybraného predikátu, přesune do bodu 4.4.2 nebo do bodu 4.4.1, kde vybere predikát jehož interpretaci bude posléze chtít měnit. V každém z těchto případu, musí průvodce pracovat již se všemi pozměněnými interpretacemi.

Na závěr této kapitoly je ještě vhodné podotknout, že jednotlivé fáze zde popsané, se mohou v zájmu uživatelského komfortu ve výsledné implementaci více či méně překrývat a také zobrazení pomocných informací bude více interaktivní než je zde možno popsat. Dále se již nebudeme zabývat konceptuálním modelem aplikace a přiblížíme si její implementační část.

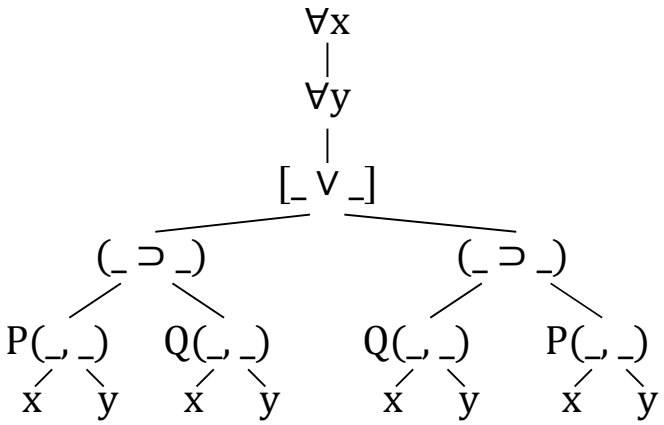
5 Základní algoritmy

5.1 Vyhodnocení pravdivosti formule v interpretaci

Nyní si rozebereme základní algoritmus aplikace pro vypočtení pravdivostní hodnoty formule v interpretaci. Algoritmus je přizpůsoben námi vymezené podmnožině formulí PL[1], nicméně jeho případné rozšíření například na formule v ne-prenexním tvaru není problém. Pokud bychom požadovali obecné řešení problému vypočtení pravdivostní hodnoty formule v interpretaci, mohli bychom tento problém převést například na posloupnost tzv. SAT problémů [7]. Tyto problémy bychom mohli následně vyřešit některým ze specializovaných programů(např. MiniSat [8]). Skutečně obecné strojové řešení problému zjištění pravdivosti formule v interpretaci v dnešní době, především z důvodu omezeného výkonu a systémových prostředků možné není.

Ještě předtím než se pustíme do popisu samotného algoritmu, je třeba navrhnout jak bude formule uložena v paměti počítače. Způsob jakým je formule utvořena přímo vybízí k použití stromové struktury, kde každý uzel této struktury představuje kvantifikátor, logickou spojku, predikát nebo funkční symbol s aritou $n > 1$. Listy tohoto stromu jsou pak konstanty nebo proměnné. Model uložení formule je blíže specifikován v kapitole 6.3.

Příklad 5.1
Uložení formule $\forall x \forall y[(P(x,y) \supset Q(x,y)) \vee (Q(x,y) \supset P(x,y))]$ do stromu:



■

Na procházení stromu formule použijeme modifikovaný algoritmus *vyhledávání do hloubky* [6]. V naší aplikaci používáme rekurzivní verzi algoritmu, která s prohledáváním začíná u kořene stromu a pokračuje vždy prvním potomkem dosud nenavštíveného uzlu. Pokud vyhledávání dospěje do bodu, kdy právě navštívený uzel je listem stromu, pokračuje se dosud nenavštíveným potomkem, předka aktuálního uzlu. Pokud již byli navštíveni všichni potomci předka aktuálního uzlu, vyhledávání pokračuje stejným způsobem o úroveň výše. Rekurzivní varianta algoritmu má tu výhodu, že není potřeba

ukládat na zásobník již navštívené uzly, "pamatování"si již navštívených uzlů se díky rekurzi děje zcela automaticky.

Jestliže během procházení stromu narazíme na predikát, vypočteme jeho pravdivostní hodnotu, kterou vrátíme o úroveň výše. K uzlům jenž jsou predikáty se tedy budeme chovat jako k listům stromu formule. Pokud bude aktuálně navštíveným uzlem jedna z logických spojek, rekurzivně vyhodnotíme její pravou a levou stranu, poté již můžeme vyhodnotit její pravdivostní hodnotu. Obdobně postupujeme i s uzlem negace v případě, že jeho potomek není predikát, kdy pouze stačí rekurzivně spočíst pravdivostní hodnotu potomka a tu posléze negovat. Jelikož prohledávání stromu probíhá od jeho kořene, je třeba se vypořádat se situací kdy narazíme na jeden z kvantifikátorů. Pokud se tak stane, tak pouze vrátíme rekurzivně spočtenou pravdivostní hodnotu potomka. Nyní již samotný algoritmus prohledávání v pseudokódu:

function *VyhodnoťFormuli(Element):*

if $Element$ **is** *Kvantifikátor***:**

return *VyhodnoťFormuli(Element* $\rightarrow$ *potomek)*

if $Element$ **is** *Predikát***:**

return $\leftarrow$ *VyhodnoťPredikát...(Element, není negovaný)*

if $Element$ **is** *Negace***:**

if $Element \rightarrow potomek$ **is** *Predikát* **:**

return *VyhodnoťPredikát...(Element* $\rightarrow$ *potomek, negovaný)*

return not *VyhodnoťFormuli(Element* $\rightarrow$ *potomek)*

if $Element$ **is** *Logická spojka***:**

$l\_pravdivost \leftarrow$ *VyhodnoťFormuli(Element* $\rightarrow$ *leváStrana)*

$p\_pravdivost \leftarrow$ *VyhodnoťFormuli(Element* $\rightarrow$ *praváStrana)*

return *VyhodnoťSpojku(*$l\_pravdivost$, $p\_pravdivost$, *Element)*

V algoritmu jsme použili dvě funkce, o kterých jsme se dosud nezmínili. Jedná se o funkce *VyhodnoťSpojku* a *VyhodnoťPredikát...*. Funkce *Vyhodnoť spojku* má za úkol vrátit pravdivostní hodnotu logické spojky. Funkce na vstupu přebírá již vypočtené pravdivostní hodnoty levé a pravé strany operátoru spolu s právě navštíveným uzlem stromu. V závislosti na typu logické spojky, který zjistíme pomocí uzlu ze vstupu funkce, jsou na pravdivostní hodnoty aplikovány operace logické spojky. Protože operátory implikace a ekvivalence v programovacím jazyce nenalezneme, musíme tyto dvě spojky převést na operace konjunkce a disjunkce. Výsledkem je tato funkce v pseudokódu:

function *VyhodnoťSpojku(PraváStrana, LeváStrana, Element):*

if $Element$ **is** *Konjunkce***:**

return *praváStrava* **and** *leváStrana*

if $Element$ **is** *Disjunkce***:**

return *praváStrava* **or** *leváStrana*

if $Element$ **is** *Implikace***:**

return not *praváStrava* **or** *leváStrana*

if $Element$ **is** *Ekvivalence***:**

return (**not** *praváStrava* **or** *leváStrana*) **and** (**not** *leváStrana* **or** *praváStrava*)

Druhá z doposud nedefinovaných funkcí *VyhodnoťPredikát...*, tvoří jádro celého algoritmu. Slouží totiž k vyhodnocení pravdivostní hodnoty predikátu v interpretaci. Při návrhu této funkce jsme přihlédli k omezením množiny formulí, která byla stanovena v kapitole 3. Z nich vyplývá, že žádný predikát nemůže obsahovat více něž dvě různé proměnné. Zdůrazněme, že k volným proměnným se budeme při vyhodnocování chovat jako by byly vázány všeobecným kvantifikátorem. Z těchto faktů budeme dále vycházet, při návrhu konkrétních funkcí pro výpočet pravdivostní hodnoty. Na základě počtu různých proměnných v predikátu, budeme vybírat jednu z funkcí *VyhodnoťPredikátJednaProměnná* nebo *VyhodnoťPredikátDvěProměnné*. Na výběr máme těchto funkcí:

function *VyhodnoťPredikátJednaProměnná(Predikát, JeNegovaný):*

- Proměnná je volná nebo vázaná všeobecným kvantifikátorem:

  for $i \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $false$ **then:**
  return $false$
  return $true$

- Proměnná vázaná existenčním kvantifikátorem:

  for $i \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $true$ **then:**
  return $true$
  return $false$

function *VyhodnoťPredikátDvěProměnné(Predikát, JeNegovaný):*

- Obě proměnné jsou volné nebo vázány všeobecným kvantifikátorem a nebo je jedna z nich vázána všeobecným kvantifikátorem a druhá je volná:

  for $i \leftarrow 1$ **to** n:
  for $j \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i), e(y_j)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $false$ **then:**
  Bylo nalezeno ohodnocení, ve kterém není predikát pravdivý a tímto je porušena podmínka všeobecných kvantifikátorů ⇒ **return** $false$
  Nebyla nalezena žádá nepravdivá valuace, tím pádem je splněna podmínka dvou všeobecných kvantifikátorů ⇒ **return** $true$

- Obě proměnné jsou vázány všeobecnými kvantifikátory:

  for $i \leftarrow 1$ **to** n:
  for $j \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i), e(y_j)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $true$ **then:**
  Byla nalezena pravdivá valuace, to postačuje ke splnění podmínky existenčních kvantifikátorů ⇒ **return** $true$
  Nebyly nalezeny žádé valuace, jenž by splňovaly podmínku dvou všeobecných kvantifikátorů ⇒ **return** $false$

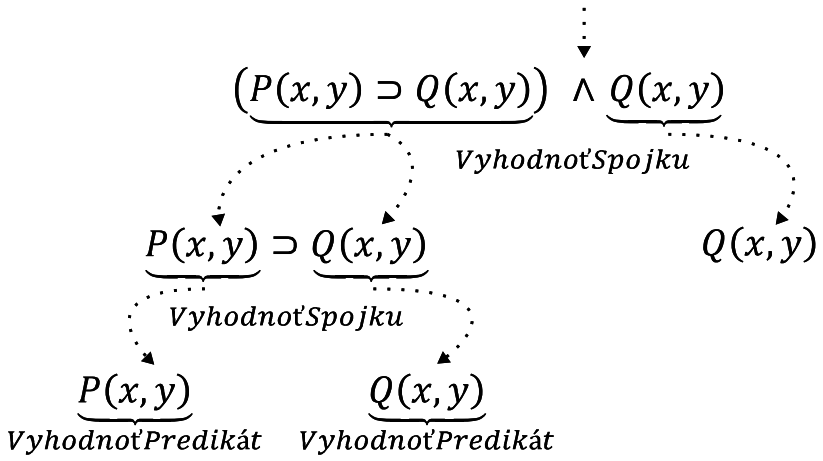
- První proměnná je vázána existenčním kvantifikátorem a druhá je buď to volná nebo vázána všeobecným kvantifikátorem:

  for $i \leftarrow 1$ **to** n:
  $pravdivost \leftarrow false$
  for $j \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i), e(y_j)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $false$ **then:**
  break
  if $pravdivost$ **is** $true$ **then:**
  Byla nalezena valuace $e(x_i)$ taková, že predikát je pravdivý pro tuto konkrétní valuaci a pro všechny valuace $e(y_1) \ldots e(y_n)$ ⇒ **return** $true$
  Nenašli jsme ani jednu valuaci pro kterou by platila podmínka kvantifikátorů ⇒ **return** $false$

- První proměnná je volná nebo vázaná všeobecným kvantifikátorem, druhá je vázána existenčním kvantifikátorem:

  for $i \leftarrow 1$ **to** n:
  $pravdivost \leftarrow false$
  for $j \leftarrow 1$ **to** n:
  $pravdivost \leftarrow$ Vyhodnoť predikát pro valuaci $e(x_i), e(y_j)$
  $pravdivost \leftarrow$ **not** $pravdivost$ **if** *JeNegovaný* **else** $pravdivost$
  if $pravdivost$ **is** $true$ **then:**
  break
  if $pravdivost$ **is** $false$ **then:**
  Nebyla nalezena žádná valuace $e(y_j)$ pro aktuální valuaci $e(x_i)$, ve které by byl predikát pravdivý $\Rightarrow$ **return** $false$
  Pro každou valuaci $e(x_1) \dots e(x_n)$ byla nalezena jedna pravdivá valuace $e(y_j) \Rightarrow$ **return** $true$

Pokud jsou argumenty predikátu pouze konstanty nebo funkce jejíž argumenty jsou konstanty, můžeme rovnou přistoupit k poslední částí algoritmu. Tou je samotný způsob zjištění pravdivostní hodnoty predikátu pro určitou valuaci. Tento problém lze jednoduše vyřešit ověřením zda vlastnosti argumentů predikátu vyhovují intepretaci predikátu. Nesmíme při tom ovšem na argumenty zapomenout aplikovat funkční symboly, pokud je daný predikát obsahuje. Samotná implementace této poslední části je značně závislá na použití programovacího jazyka, proto zde konkrétní způsob řešení nebudeme rozebírat. Na obrázku 9 je vidět, že predikát $Q(x, y)$ se ve formuli vyskytuje hned několikrát. Algoritmus můžeme dále optimalizovat, tím že implementujeme mechanismus, který si bude pamatovat pravdivostní hodnoty všech již vyhodnocených predikátů.



Obrázek 9: Procházení formule algoritmem

5.2 Průvodce interpretací

Dále se podíváme na algoritmus průvodce interpretací. Pří návrhu tohoto algoritmu jsme vycházeli z faktu, že formule, jenž bude předmětem zkoumání pravdivosti, je uložena ve stromové struktuře. Proto, stejně jako v předchozí části použijeme na prohledání grafu formule upravený algoritmus *vyhledávání do hloubky* [6]. Dále bylo při návrhu přihlédnuto k faktu, že v době kdy je algoritmus průvodce na formuli aplikován, jsou již známy všechny pravdivostní hodnoty doposud interpretovaných predikátů a proto bude algoritmus na formuli nahlížet jako by se jednalo o formuli výrokové logiky.

Možná Vás může nyní napadnout, proč jednoduše ke zjištění pravdivosti formule v interpretaci nepoužijeme algoritmus uvedený v předchozí kapitole? Důvod je zcela prozaický. Algoritmus uvedený uvedený v kapitole 5.1 vyžaduje pro zjištění pravdivosti formule v interpretaci, aby všechny predikáty a funkční symboly byly interpretovány. Existuje ovšem mnoho situací, kdy není třeba mít interpretovánu celou formuli, abychom mohli zcela jistě říci, že daná interpretační struktura již nemůže být modelem této formule. Nebylo by lepší uživatele na chybu při interpretaci upozornit ihned, než jej zbytečně nutit interpretovat všechny funkční symboly a predikáty v situaci kdy daná interpretační struktura již stejně být modelem formule nemůže? Právě k tomuto účelu slouží algoritmus průvodce, který dokáže z částečně interpretované formule vyvodit, zda může ještě uživatel dosáhnout správného řešení.

Nyní však již bez zbytečného zdržování přistupme k vysvětlení algoritmu samotného. Jako první si rozebereme funkci *JeFormuleVyhovující*, jenž je definována takto:

function *JeFormuleVyhovující(formule):*

možnéVýsledky ← *ZjistiVýsledky(formule → kořen, žádnáStrana)*

return *1* **in** *možnéVýsledky*

Jejím základním úkolem je pomocí další funkce *ZjistiVýsledky* zjistit jakých možných pravdivostních hodnot může nabývat formule. Poté funkce ověří, zda se v možných výsledcích nachází kladná pravdivostní hodnota, z čehož plyne, že formule může ještě v dané částečné interpretaci být pravdivá. Jestliže je již interpretační struktura kompletní, pak se nejedná o nic jiného než o ověření zda je interpretační struktura modelem formule. Je dobré poznamenat, že funkce *ZjistiVýsledky* může na nákladě stavu a správnosti interpretace vracet tyto tři různé výsledky: {0}, {1}, {0,1}. Je tedy jasné, že v případě, kdy tato funkce vrátí hodnotu {0}, tak daná interpretační struktura je nevyhovující. Naopak v případě návratu {1} nebo {0,1} můžeme usoudit, že interpretační struktura je prozatím vyhovující. Následně si definujeme funkci *ZjistiVýsledky*:

function *ZjistiVýsledky(element, stranaOperátoru):*

if *element* **is** *logickáSpojka* **:**

leváStrana ← *ZjistiVýsledky(element → levýPotomek, leváStrana)*

praváStrana ← *ZjistiVýsledky(element → pravýPotomek, praváStrana)*

možnéVýsledky ← *AplikujSpojku(leváStrana, praváStrana, element → typSpojky)*

UložTabulkuDoStromu(leváStrana, praváStrana, element)

return *možnéVýsledky*

if (*element* **is** *Negace*) **and** (*element* → *Potomek* **is not** *Predikát*)**:**

možnéVýsledky ← *ZjistiVýsledky(element → potomek, žádnáStrana)*

UložNegaciDoStromu(element)

return not all of *možnéVýsledky*

if (*element* **is** *Predikát*) **or** ((*element* **is** *Negace*) **and** (*element* → *Potomek* **is** *Predikát*))**:**

- pravdivostní hodnota predikátu je již známa:
  možnéVýsledky ← *VytořStranuTabulky(element → pravdivostníHodnota, stranaOperátoru)*
- pravdivostní hodnota predikátu není známa:
  možnéVýsledky ← *VytořStranuTabulky({0, 1}, stranaOperátoru)*

return *možnéVýsledky*

Úkolem této funkce je prohledání stromu formule, zjištění možných pravdivostních hodnot jednotlivých podstromů a následné aplikování logických spojek na tyto hodnoty. Vidíme, že funkce se větví na tři části, které si nyní samostatně rozebereme:

- První větev funkce řeší případ, kdy během prohledáváni stromu formule narazíme na logickou spojku. Funkce tedy jednoduše rekurzivně zjistí množné hodnoty, kterých může nabývat levá a pravá strana operátoru. Funkce dostane tyto hodnoty v podobě části pravdivostní tabulky, v závislosti na tom o kterou stranu operátoru se jedná. Na takto získanou pravdivostní tabulku je poté aplikována další funkce *AplikujSpojku*, která pouze aplikuje logickou spojku na danou pravdivostní tabulku. Tato funkce opět vrací pouze možné pravdivostní hodnoty, kterých může celá logická spojka nabývat tedy $\{0\}$, $\{1\}$ nebo $\{0,1\}$. Hodnotu, kterou jsme získali jako výstup funkce *AplikujSpojku* poté předáme v rekurzi o úroveň výše.
- Druhá větev funkce zahrnuje situaci, jenž nastane v případě že v rekurzivním prohledávání narazíme na negaci. Stejně jako předchozí větev i tato nejdříve rekurzivně zjistí jakých pravdivostních hodnot může nabývat její operand. Takto získané hodnoty poté neguje a opět vrací o úroveň výše. Stojí zato podotknout, že tato větev ztrácí význam v případě, že navrácená množina možných pravdivostních hodnot je $\{0, 1\}$. I po aplikaci negace na prvky této množiny zůstává množina nezměněna.

- Poslední větev funkce se zaobírá případem, kdy během prohledávání stromu narazíme na predikát. V této situaci je pouze potřeba s pomocí funkce *VytořStranuTabulky* sestavit část pravdivostní tabulky pro danou stranu operátoru. Jestliže je pravdivostní hodnota predikátu v již známa, tak pro konstrukci části tabulky použijeme právě tuto hodnotu. V opačném případě použijeme všechny hodnoty, kterých může predikát nabývat, tedy $\{0, 1\}$.

Jak celá to funkce pracuje, si můžeme prohlédnout na obrázku 10. Schválně jsme zde použili poněkud složitější formuli, aby bylo názorně vidět, že o pravdivosti formule v dané interpretaci můžeme rozhodnout již v polovině interpretačního procesu. Než se posuneme dále, řekneme si ještě něco málo o dvou dosud nepopsaných funkcích *UložTabulkuDoStromu* a *UložNegaciDoStromu*. Tyto dvě funkce slouží k vytvoření podobného stromu pravdivostních tabulek a negací jaký můžeme vidět na obrázku 10. Abychom mohli uživateli vůbec zobrazit informaci o tom, které predikáty jsou špatně interpretovány v situaci, kdy uživatel měnil interpretaci funkčního symbolu nebo predikátu vyskytujícím se na více místech formule, musíme si uložit všechny pravdivostní tabulky vypočtené algoritmem. Pokud máme všechny pravdivostní tabulky takto uloženy, stačí se pouze podívat, zda pravdivostní hodnota predikátu je v přípustné množině správných hodnot, kterou můžeme vyčíst právě z takto uložené tabulky. Samotnou implementaci uložení tabulek zde z důvodu omezeného prostoru rozebírat nebudeme.

V případě, že uživatel nemění interpretace funkčních symbolů nebo se aktuálně vybraný predikát nevyskytuje na více místech, není třeba vypočtené tabulky nijak evidovat, vystačíme si pouze se znalostí aktuálně vybraného predikátu. Pokud jsme v této situaci dostali od algoritmu průvodce odpověď, že daná částečná interpretace již nemůže být nikdy modelem formule, můžeme spolehlivě usoudit, že problém je v interpretaci aktuálně vybraného predikátu.

Dále následují definice funkcí pro aplikování logické spojky na části pravdivostní tabulky a také funkce pro vytvoření částí těchto tabulek. Tyto čtyři funkce jsou v zásadě velmi jednoduché. Jejich význam lze velmi jednoduše odvodit z názvu a proto již dále nebudeme popisovat.

```
function AplikujSpojku(leváStrana, praváStrana, spojka):
    výsledek ← {}
    for i ← 0 to 3 :
        if leváStrana[i] or praváStrana[i] is - :
            continue
        výsledekOperace ← aplikuj sojku na leváStrana[i], praváStrana[i]
        if operace not in výsledek :
            add operace to výsledek
```

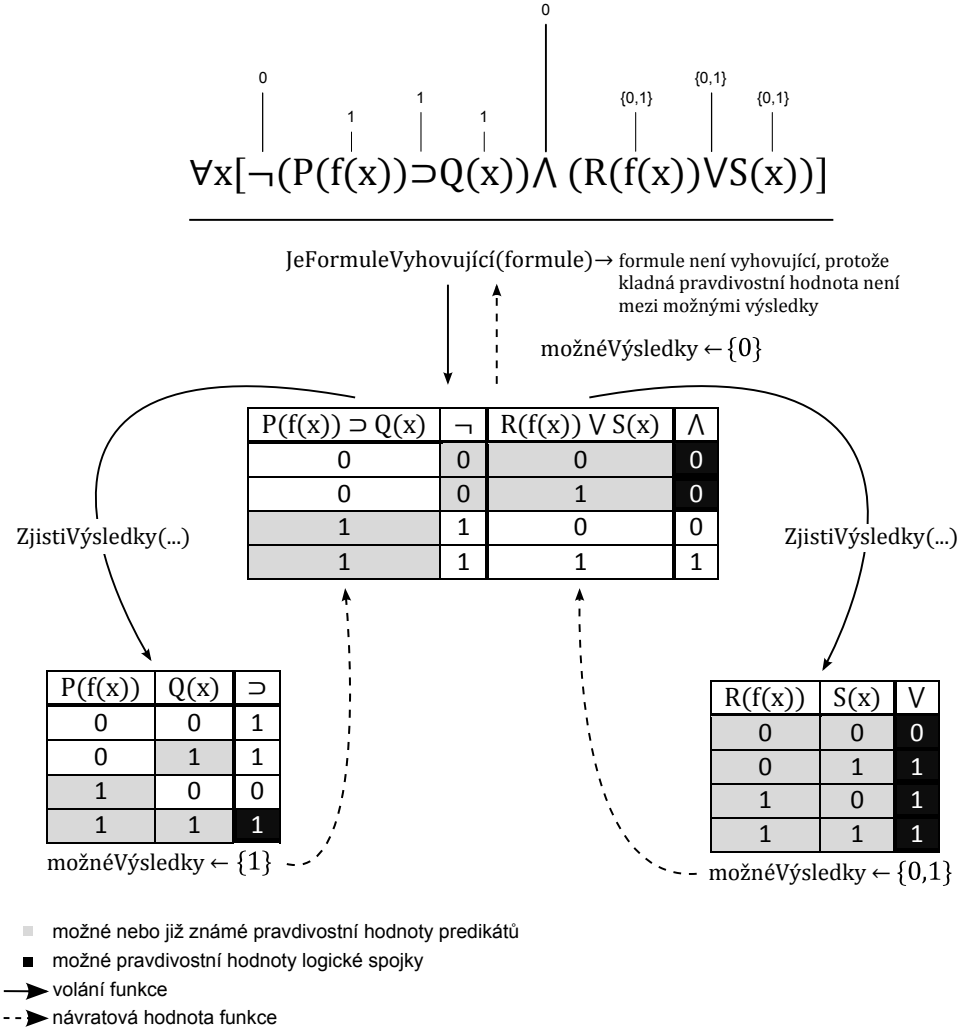
```
    return výsledek

function VytvořStranuTabulky(platnéHodnoty, stranaOperátoru):
    if stranaOperátoru is žádnáStrana:
        return platnéHodnoty
    if stranaOperátoru is leváStrana:
        return VytvořLevouStranuTabulky(platnéHodnoty)
    if stranaOperátoru is praváStrana:
        VytvořPravoutranuTabulky(platnéHodnoty)

function VytvořLevouStranuTabulky(platnéHodnoty):
    if only 1 in pravdovostníHodnoty :
        return { -, 1, -, 1 }
    if only 0 in pravdovostníHodnoty :
        return { 0, -, 0, - }
    if 0 and 1 in pravdovostníHodnoty :
        return { 0, 1, 0, 1 }

function VytvořPravouStranuTabulky(platnéHodnoty):
    if only 1 in pravdovostníHodnoty :
        return { -, -, 1, 1 }
    if only 0 in pravdovostníHodnoty :
        return { 0, 0, -, - }
    if 0 and 1 in pravdovostníHodnoty :
        return { 0, 0, 1, 1 }
```

Nyní bychom měli mít dostatečný přehled o celé aplikaci a algoritmech v ní použitých, můžeme tedy pokračovat předposlední kapitolou, ve které si rozebereme implementaci hlavních částí aplikace na úrovni jazyka UML.



Obrázek 10: Funkce algoritmu průvodce na složitější formuli

6 Konceptuální model aplikace

6.1 Architektura aplikace

Při návrhu architektury aplikace jsme vycházeli z kapitoly 4, kde jsme se seznámili s jejími hlavními funkcemi. Tyto jednotlivé funkce jsme poté v podobě subsystémů promítli do globální architektury aplikace, jak můžeme vidět na obrázku 11. V knize *Code Complete* [4] se můžeme v 5. kapitole dočíst, že *"systémový diagram by měl být acyklickým grafem."* Architektura naší aplikace se snaží tohoto cíle dosáhnout udržováním tzv. **volných vazeb** (loose coupling) mezi jednotlivými subsystémy. Volných vazeb mezi subsystémy lze velmi jednoduše dosáhnout aplikováním známého návrhového vzoru **Facade** [5], kdy veškerá komunikace subsystému s okolím probíhá skrze pevně definované rozhraní jediné třídy. Vlastní komunikaci mezi jednotlivými subsystémy skrze rozhraní definovaná fasádou má na starost další subsystém s názvem **AppManager**, který zároveň zpracovává i požadavky vedené z uživatelského. Je tedy jasné, že v takto navržené architektuře případná záměna jednoho ze subsystémů neovlivní chování subsystémů ostatních, za předpokladu že data poskytnuta změněným systémem budou korektní. Celý návrh architektury není nepodobný jednomu z návrhových vzorů zvanému **Mediator** [5].

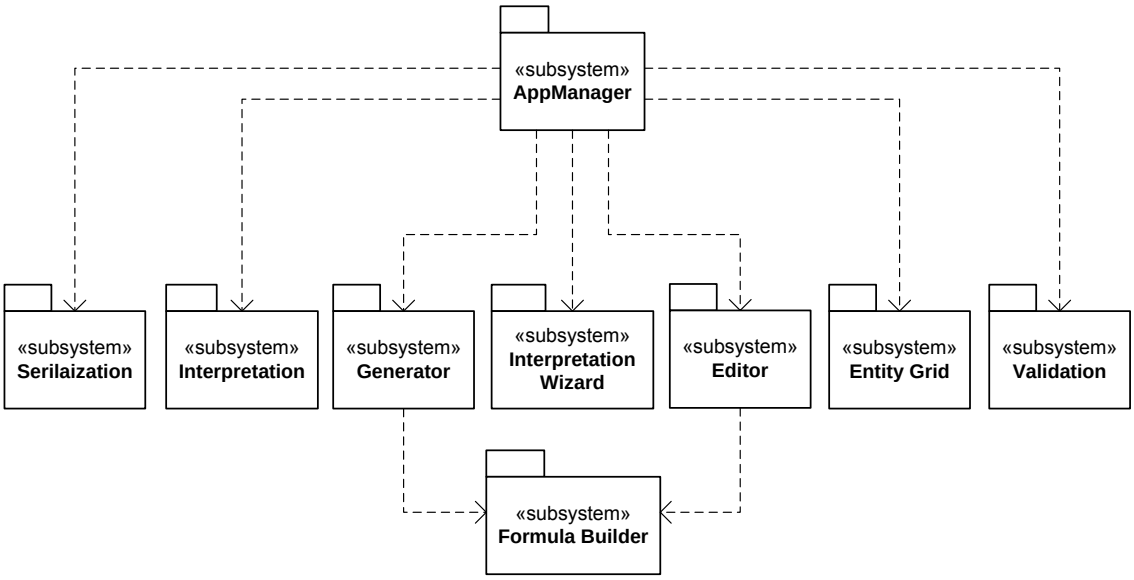
Jak již z předchozích kapitol víme, výsledná aplikace musí podporovat dvojí způsob vkládání formulí, pomocí vizuálního editoru a generátoru náhodných formulí. Proto jsme v duchu principu DRY pro vytváření objektového stromu formulí vytvořili podsystém **Formula Builder**. Úkolem systémů editoru a generování formulí je tedy pouze poskytnutí vlastního rozhraní mezi zbytkem aplikace a samotným vytvářením formulí.

V následujících částech této kapitoly si rozebereme implementaci generátoru formulí a uložení formulí do stromové struktury. Ostatním podsystémům jako je persistentní uložení formulí či 3D zobrazení universa se věnovat nebudeme, neboť jejich implementace je úzce svázána s konkrétním programovacím jazykem a dalšími použitými technologiemi. Celá tato kapitola by nám i přesto měla poskytnout dobrý základ k tomu, abychom mohli aplikaci implementovat v kterémkoliv jazyce podporujícím OOP. Pro naši vzorovou implementaci byl vybrán jazyk C# na platformě .NET s použitím technologie Windows Presentation Foundation.

6.2 Generátor formulí

V kapitole 3 jsme pro vymezení podmnožiny formulí PL^1 užili bezkontextovou gramatiku, která bude tvořit základ generátoru formulí. Nicméně nic nám nebrání v tom, abychom jí využili i ve vizuálním editoru formulí pro kontrolu vstupů uživatele. Bezkontextová gramatika je však spíše určena ke generování bezkontextových jazyků a k tomuto účelu jí také užijeme. Celý problém náhodného generování formulí se právě díky této gramatice, převede na pouhé náhodné vybrání pravidla neterminálu. Náhodné vybírání pravidel ovšem musíme implementovat takovým způsobem, abychom generovali pouze správně utvořené formule PL^1, jak je popsáno na konci kapitoly 3.

Pro implementaci bezkontextové gramatiky využijeme návrhový vzor známý jako **Interpretor**. Tento návrhový vzor ovšem ve své původní podobě, tak jak je popsán v



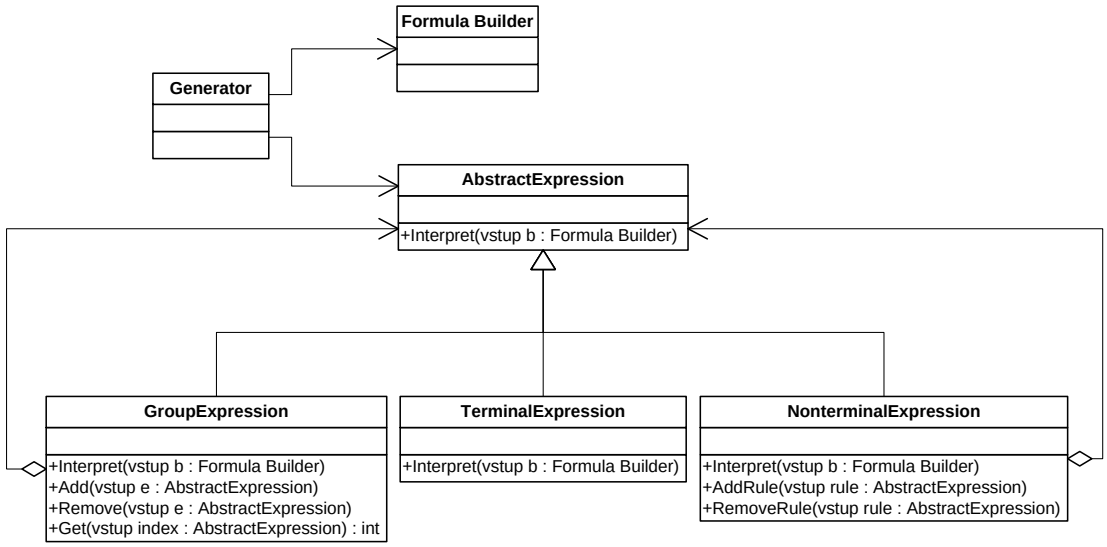
Obrázek 11: Architektura aplikace

původní knize *Design Patterns - Elements of Reusable Object-Oriented Software* [5], neumožňuje definici pravidla složeného z několika terminálů a neterminálů. Proto si jej pro použití v naší aplikaci mírně upravíme, přidáním podpory pro tyto složená pravidla. Výsledný návrh můžeme vidět na diagramu obrázku 12. Je dobré si všimnout, že všechny části gramatiky mají sdílený kontext *Formula Builder*. Jedná právě o již dříve zmiňovaný podsystém, který je zodpovědný za samotné utváření stromu formule. Objekty *TerminalExpression* poté pouze volají vhodné metody FormulaBuilderu, jenž následně přidají do stromu formule vhodný element. Instance třídy *TerminalExpression* představující proměnnou s názvem *x*, může tedy například způsobit volání metody *AddVariable("x")*.

6.3 Uložení formulí

Jestliže již tušíme jak probíhá utváření formulí na úrovni generátoru, je dobré si následně utvořit představu, jakým způsobem jsou tyto formule uloženy v paměti. Víme, že celá formule bude uložena ve stromové struktuře. Co však ještě nevíme, je jak bude tato struktura vypadat na úrovni jednotlivých tříd. Celou strukturu je možno shlédnout na diagramu obrázku 13.

Jedná se o rozšíření dobře známého návrhového vzoru **Composit** [5], do něhož jsme pouze přidali další typy listových položek (*Constant*, *Variable*) a uzlových položek (*Predicate*, *Negation*, *Quantificator*, ...). Za povšimnutí stojí třídy *ConstantType*, *FunctionType* a *PredicateType* představující typy jednotlivých symbolů formule. Instance těchto tříd jsou poté sdíleny všemi objekty stromové struktury, představující daný typ se stejným jménem. To oceníme především ve chvíli, kdy formule obsahuje stejné symboly, jenž se vyskytují na více místech. Dále tyto třídy uchovávají informace potřebné k interpretaci



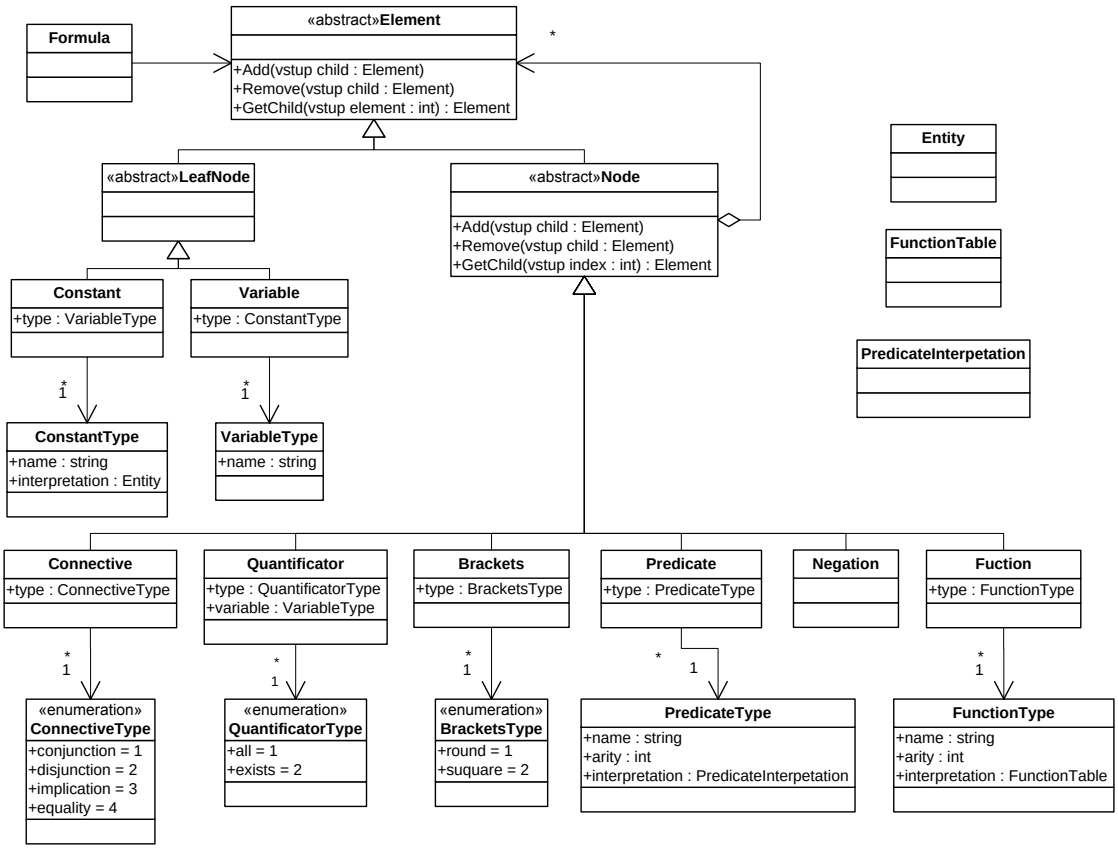
Obrázek 12: Gramatika generátoru

jednotlivých symbolů, jak můžeme vidět například u třídy *ConstantType*. Atribut *interpretation* této třídy, skrze objekt typu *Entity*, představuje právě jedno individuum universa.

V diagramu uložení formulí si dále všimněme tříd *Entity*, *FunctionTable* a *PredicateInterpretation*. Právě z instancí těchto tříd sestává interpretace jednotlivých funkčních symbolů a predikátů. Instance třídy *Entity* představují právě jedno individuum universa s vlastnostmi uvedenými v kapitole 4.3, jedná se tedy o interpretaci konstanty. Následující třída *FunctionTable* je implementací interpretace funkčního zobrazení. Musí tedy splňovat všechny předpoklady, jenž pro takové zobrazení platí, tedy jednoznačnost zobrazení zprava a totalitu pro dané universum. V případě interpretací predikátů hovoříme o objektech typu *PredicateInterpretation*, pro které je nutno pomocí operátorů programovacího jazyka a dalších pomocných metod implementovat jednotlivé relace a podmnožiny na universu v souladu s výčtem interpretací pro predikáty uvedeným v kapitole 4.3. Implementace jednotlivých interpretačních tříd se může v závislosti na programovacím jazyce velmi lišit proto zůstaneme pouze u jejich obecného popisu, nicméně si pro ujasnění na výpisu 1 demonstrujeme alespoň část implementace třídy *PredicateInterpretation* v jazyce C# pro interpretaci "x je nad nebo pod y."

```
public class PredicateInterpretation
{
    public void IsValid(Entity x, Entity y)
    {
        return (x.PositionY > y.PositionY) || (x.PositionY < x.PositionY)
    }
}
```

Výpis 1: Implementace relace v C#



Obrázek 13: Uložení formule

7 Závěr

Výsledkem této práce je aplikace s názvem *FOLModeler*, kterou je možno nalézt na přiloženém CD. Díky této aplikaci mohou zájemci o studium predikátové logiky prvního řádu snáze a lépe porozumět jejím základům, které mohou následně rozšířit o další poznatky, jenž je možno dále aplikovat například v oblasti expertních systémů. Pokud bychom dosud v této oblasti hledali kvalitní výukový software, který umožňuje přehledné zobrazení relací mezi jednotlivými individui universa, pohodlnou práci s formulemi a jejich následnou interpretací, museli bychom sáhnout po již zmíněné aplikaci *Tarski's World*, jejíž stav však není úplně vyhovující. Při psaní aplikace *FOLModeler* jsme se tedy poučili z chyb *Tarski's World* a dále doplnili několik funkcí, které v původní aplikaci tolik chyběly. Zejména bychom měli vyzvednout vizuální editor formulí, díky němuž je student oproštěn od všech problémů spojených nekorektními vstupy do aplikace a může se tak plně soustředit na strukturu samotné formule. Dále uveďme systém průvodce interpretací, který umožňuje jedinečným způsobem uživatele postupně navést ke správnému interpretování formule. Nesmíme také zapomenout na systémy náhodného generování formulí a universa, jenž vnáší do aplikace velkou míru variability a také trojrozměrné zobrazení celého universa.

Nicméně i přesto, že aplikace *FOLModeler* obsahuje pro běžného uživatele dostatečné množství funkcí, nabízí se hned několik oblastí kde by v budoucnu mohlo dojít k jejímu rozšíření. Průvodce interpretací může být doplněn o heuristické algoritmy vylepšující nápovědu. Dále se nabízí rozšíření množiny formulí o formule v ne-prenexním tvaru, vizualizace průběhu vyhodnocení pravdivosti formule v interpretaci, což je zejména vhodné ve spojení s kvantifikátory. V neposlední řadě můžeme také přidat další typ universa nebo i funkce pro automatické nalezení modelu formule či generování testových otázek. Vzhledem k rozsahu práce však nemohly být všechny tyto funkce do aplikace nyní zahrnuty.

Výsledky této práce budou následně využity v projektu CZ.1.07/2.2.00/07.0217, ORGANON – Learning Management Systém (LMS) pro výuku logiky.

8 Literatura

[1] BARKER-PLUMMER, Dave; BARWISE, Jon; ETCHEMENDY, John. Tarski's World : Revised and Expanded : University of Chicago press, 2007. 144 s. ISBN 1575864843.

[2] DUŽÍ, Marie. Matematická logika. Ostrava, 2003. 131 s. Skriptum. VŠB-TUO.

[3] MENŠÍK, Marek; Jarotek, Vladimír; Číhalová, Martina. Interpretace formulí v PL1 do přirozeného jazyka: In Organon VI, 2009

[4] MCCONNELL , Steve. Dokonalý kód : Umění programování a techniky tvorby software. Brno : Computer Press, 2006. 896 s.

[5] GAMMA, Erich, et al. Design Patterns : Elements of Reusable Object-Oriented Software. : Addison-Wesley Pub Co, 1994. 416 s.

[6] Depth-first search In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-03-31]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Depth-first_search>.

[7] Boolean satisfiability problem In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, , [cit. 2010-03-31]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Boolean_satisfiability_problem>.

[8] [online]. [cit. 2010-03-31]. MiniSat. Dostupné z WWW: <http://www.minisat.se>.

9 Přílohy

9.1 Obsah přiloženého CD

- *Install* - instalační balíček aplikace FOLModeler
- *Zdrojové kódy*
  - *FOLModeler* - zdrojové kódy aplikace FOLModeler
  - *LaTeX* - zdrojové kódy práce
- *3D vizualizace modelů PL1.pdf* - samotná bakalářská práce